Security Models and Graphical Representation of Security

Afzul Mohmud

BSc. Computing

2002/2003

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student) _____

Summary

Formal security models are difficult concepts to learn about. Security models are usually described in a very mathematical nature, which makes them very precise, but can also make them hard to understand. This project aimed to produce a piece of software, which would demonstrate the use of graphical metaphors to represent a security model. It was hoped that the resulting software would make it easier for people to learn about security models. In order to achieve the aim of this project, several security models had to be studied, metaphors had to be designed and a functional piece of graphical software had to be designed and implemented.

Project Objectives

The objectives agreed for this project were:

1. To carry out research to investigate various security models and their features.
2. To demonstrate an understanding of how these security models work and differ.
3. To investigate the features of SQL Server 2000's security model and to compare these to the features of the security models investigated for objective 1.
4. To research available programming languages for implementing a software tool to represent a security model graphically.
5. To research appropriate development methodologies that could be used to develop the software.
6. To design graphical metaphor(s) based on the security models investigated
7. To design a piece of software that demonstrates a security model using the graphical metaphors designed previously.
8. To implement, test and evaluate this software.

Overview

Chapter 1 gives a brief introduction to the project and defines the minimum requirements and the evaluation criteria. Chapter 2 highlights some of the planning done for the project, including, the development methodology and language chosen as well as detailing some of the project management carried out. Chapter 3 analyses several formal security models as well as a real life security model i.e. SQL Server 2000's security model. Several comparisons are also presented in Chapter 3. Chapter 5 presents the designs for some metaphors to represent two formal security models. Chapters 5, 6 and 7 detail some of the code design and implementation details and also describe the testing carried out. Finally in Chapter 8, the findings of the evaluation process are presented and some future directions for the work are discussed.

Acknowledgements

Thanks to Dr Stuart Roberts for the continued help and advice he has provided throughout the course of the project. Also thanks to Bill Whyte for some useful ideas and feedback.

Thanks to all the people who took part in evaluating the software and also thanks to anyone who provided any feedback about the software.

Finally thanks to my family and friends, for the support and help they have offered throughout the project and the degree programme.

Contents

Summary……...……i

Acknowledgments……...ii

Contents……...……iii

1. Introduction……...…1

1.1. Motivation and Aim of the Project……..…1

1.2. Minimum Requirements……..…1

1.3. Evaluation Criteria……...1

2. Project Planning……..…3

2.1. Introduction……..…3

2.2. Methodologies for Software Development…….3

2.2.1. The Waterfall Model……3

2.2.2. The Waterfall Model with Feedback……3

2.2.3. Choice of Methodology……3

2.3. Development Languages…….4

2.3.1. Java3D……..4

2.3.2. OpenGL……4

2.3.2.1. OpenGL Toolkits…….5

2.3.3. Choice of Development Language……...5

2.4. Project Milestones……...6

2.5. Summary……..……...6

3. Background Research……...7

3.1. Introduction…….7

3.2. Analysis of Security Models……...7

3.2.1. The Bell-LaPadula Model……..8

3.2.2. The Harrison-Ruzzo-Ullman Model……10

3.2.3. The Chinese Wall Model…….12

3.2.4. The Role Based Access Control Model……...14

3.2.5. Comparison of the Security Models……18

3.3. SQL Server 2000's Security Model……...20

3.3.1. Analysis of SQL Server 2000's Model……20

3.3.2. Comparison to Other Security Models……21

3.4. Summary……..……...21

4. Metaphor Design……22
4.1. Introduction……22
4.2. What Makes A Good Graphical Metaphor? ……22
4.3. Methodology for Designing A Graphical Metaphor……22
4.4. Mapping Security Models Onto Metaphors……24
4.4.1. The Bell-LaPadula Model……24
4.4.1.1. Main Concepts Revisited……24
4.4.1.2. Designing the Metaphor……25
4.4.2. The Role Based Access Control Model……28
4.4.2.1. Main Concepts Revisited……29
4.4.2.2. Designing the Metaphor……29
4.4.3. Applicability of the Metaphors to Other Security Models……35
4.5. Summary……36

5. Interface Design……37
5.1. Introduction……37
5.2. The Bell-LaPadula Model……37
5.2.1. Functionality of the Software……37
5.2.2. User Controls Required……38
5.2.3. Integrating the User Controls with the Metaphor……38
5.3. The Role Based Access Control Model……39
5.3.1. Functionality of the Software……39
5.3.2. User Controls Required……40
5.3.3. Integrating the User Controls with the Metaphor……41
5.4. Summary……42

6. Code Design And Implementation……43
6.1. Introduction……43
6.2. The Bell-LaPadula Model……43
6.2.1. Class Design……43
6.2.2. Reading From Files……43
6.2.3. Drawing Functions……44
6.3. The Role Based Access Control Model……45
6.3.1. Class Design……45
6.3.2. Reading From Files……46
6.3.3. Drawing Functions……46
6.4. Summary……47

7. Implementation and Testing……48
7.1. Introduction……48
7.2. Implementation Issues……....48
7.3. Testing……..……..49
7.4. Important Information……49

8. Evaluation……..50
8.1. Introduction…..……...50
8.2. Results of Applying Evaluation Criteria……50
8.2.1. Evaluating the Objectives and Minimum Requirements……..50
8.2.2. Evaluating the Software……..……..51
8.2.3. Evaluation of User Feedback……..52
8.3. Directions for Future Work……55
8.4. Overall Conclusions……....55

References……..57

Appendix A – Personal Reflection……..59

Appendix B – Project Schedules…...……..60

Appendix C – Potential Design For RBAC Software..……..63

Appendix D – Test Plans……..65

Appendix E – Software Documentation……66

Chapter 1 – Introduction

1.1 Motivation and Aim of the Project

When attempting to understand computer security models, people are often exposed to a vast amount of formal and mathematical detail and little visual representation. This can make it difficult to understand what the models represent and what the properties of each model mean. It is often difficult to visualise security models and this can also make it harder for people to understand them. Therefore, this project aimed to simplify the complexity of security models by representing them graphically.

The aim of this project was to investigate different models for computer security and to produce a piece of software, which demonstrated one or more of these models through the use of graphical metaphors. Through this, the goal was to determine whether security can be represented adequately through the use of graphical metaphors and also to determine how much these metaphors can aid in the understanding of the structure and the features of a security model.

1.2 Minimum Requirements

The minimum requirements agreed for this project were:

1. To investigate at least two different computer security models.
2. To compare these security models with SQL Server 2000's security model.
3. To devise a suitable graphical metaphor to represent one of these security models.
4. To design and develop software that implements at least one graphical metaphor.
5. To test and evaluate the software that has been produced

1.3 Evaluation Criteria

Upon the completion of the project, the success of the project was to be evaluated. To evaluate the project, a set of evaluation criteria were defined. These were: -

1. Evaluating the Objectives and Minimum Requirements

- **Has the project achieved the objectives that have been set out?**
  This criterion will evaluate the success of the project in relation to the objectives that were set out at the start of the project.

- **Has the project achieved the minimum requirements or have they been exceeded?**
  This criterion will evaluate whether the project has actually achieved the minimum requirements that were set out and whether or not these were exceeded in any way.

2. Evaluating the Software

- **Was the choice of programming language and development methodology appropriate?**
  This criterion will evaluate whether the programming language and methodology chosen for the project were appropriate i.e. did they work well, or cause any problems etc.

- **Is the software neatly coded and extensible?**
  This criterion will evaluate whether the software code is understandable and whether it will be modifiable (if necessary) in the future.

- **Does the software work and provide enough functionality?**
  This criterion will evaluate whether the software has any major bugs. It will also evaluate whether the software does enough to represent the security model.

3. Evaluation of User Feedback

- **Does the software help people to understand the security model?**
  This criterion will evaluate how well the software works in terms of helping people to learn about security models. This is important, as it will determine whether or not the metaphors that have been developed are serving their purpose i.e. simplifying the security models. To carry out this step of the evaluation, users will be given an overview of the security model. Then the users will be given a brief explanation of the metaphor and the software, and will be allowed to use the software for a short amount of time. After this the users will be asked if the software aided, increased or hindered their understanding of the security model. This criterion will be evaluated by the response received from users after this experiment.

- **Can users correctly identify what the software/metaphors represent and what is happening as the software runs?**
  This criterion will evaluate how well users can actually interpret what is going on in the software. This is important, as it will help to determine, not only whether users understand the metaphors, but also whether they can identify what each change to the metaphor means. To carry this out, users will be shown a sequence of events occurring in the software representing various concepts of the security model, i.e. violation of rules. The user will then be asked to identify what has occurred. This criterion will be evaluated by the results of these tests.

- **Do users think the software is easy to use?**
  This criterion will evaluate the general usability of the software. This will be evaluated by getting some general feedback from the users about the interface and how easily they understood things. This will help to identify whether the interface provided is appropriate and whether or not the software in general is easy to use and understand.

Chapter 2 – Planning

2.1 Introduction

In this chapter various elements of planning are done to plan the way the project will be carried out. Various software development methodologies and programming tools are investigated and a choice is made as to which methodology and programming language to use. A schedule for completion of the project and milestones that must be achieved during the project are also defined in this chapter.

2.2 Methodologies for Software Development

In order to implement software displaying security models using graphical metaphors, a structured development model had to be used. This model would allow the project to be planned out according to a certain structure and would allow the project to be scheduled so that the work would be well spread over the course of the project. Two widely used development models for small projects are the waterfall model and the waterfall model with feedback. In this section, these models are analyzed and the justification is presented for the methodology that was used for this project.

2.2.1 The Waterfall Model

The Waterfall model is sometimes known as the "classical model of system development" [1]. It is a five-stage model that identifies the stages that must be performed and the order in which they must occur. It is also known as a "one shot" or "once through" model, [1], as it relies on everything being right the first time round and provides limited scope for going back and changing things. According to Jesty [2], this is most suited to "short, low risk projects". A major disadvantage of this model is that the product is not available until the end of the project [2].

2.2.2 The Waterfall Model with Feedback

The waterfall model with feedback is "a modified version of the waterfall model" [1]. This model allows feedback between each stage. This leads to a model where an iterative approach is used. This model allows changes to previous stages and allows "a more incremental delivery of the final product, addressing some of the major issues of the simple waterfall model" [2].

2.2.3 Choice of Methodology

For this project an iterative methodology was used. This meant that the methodology followed the stages of the "waterfall model with feedback". This was chosen ahead of the simple waterfall model as it allowed development in iterations. This meant that the software would be developed in an incremental fashion, which allowed the software to be evaluated after each iteration. This evaluation could then be taken into account during the next design phase. This approach, as mentioned in [1], is

well suited to projects where there are frequent deadlines and meetings where the progress of the project has to be demonstrated, as in a final year project. With the simple waterfall model, there wouldn't be a product to show in these meetings, as the development would be in the middle of a single stage. Whereas with the iterative approach, several iterations may have been completed and there would be something to demonstrate at each meeting.

2.3 Development Languages

When developing software, a choice needs to be made as to which programming language the software should be written in. For this project the programming language needed to support graphical displays, as the software would need to show graphical metaphors. In this section the programming languages available for this project are analysed and the final choice of language is presented.

2.3.1 Java3D

Java 3D is an extension to the Java programming language. "Java 3D is a network-centric, scene graph-based API, that revolutionizes 3D graphics application development"[3]. This means that Java 3D is based around scene graphs. Therefore, the developer works with the scene, rather than with the low level geometry. This means there is more focus on composition.

Java 3D is a set of Java class libraries "designed to be used in conjunction with the core libraries of the Java 2 specification"[4]. This means that programs written in Java3D can make use of many of the other facilities provided by the standard Java classes. Java3D can also support and manipulate many different file formats. It also supports many of the standard graphics features such as textures, lighting, animation, different views etc. [4]. The primary aims of Java 3D (adapted from [3]) are: -

- To provide cross-platform integration by making use of the Java Virtual Machine mechanism so that the same program can potentially run on any machine.
- To provide, through the use of simplistic scene graphs, a rich set of features for creating interesting 3D applications without having to write complex graphical code.
- To provide performance scalability, i.e. to allow programs to be created which will scale across different environments and different architectures.

2.3.2 OpenGL

"OpenGL is a software interface to graphics hardware which consists of around 150 distinct commands that are used to specify the objects and operations needed to produce interactive three-dimensional applications..... OpenGL is designed as a hardware-independent interface to be implemented on many different hardware platforms." [5]

OpenGL provides the facility to build up a model from geometric primitives such as points, lines and polygons. OpenGL can be used to create complex 2D or 3D scenes. It can create complex shapes and objects and arrange them in three-dimensional space. There is also the ability to change the viewpoint of the scene. OpenGL allows objects to be lit using simulated light sources and even using pre-loaded textures to achieve a more realistic look. It is also possible to achieve animation using OpenGL as the center point of an object can be moved around and then the object can be drawn relative to this point.

2.3.2.1 OpenGL Toolkits

OpenGL doesn't include an API to perform windowing tasks or to obtain user input and it doesn't have any commands for drawing three-dimensional objects such as cubes or spheres. However the OpenGL Utility Library (GLU) provides modeling features such as quadric surfaces. There are also other libraries, such as the GLUT library.

The GLUT library was developed "as a cross platform library for use with OpenGL" [6]. It provides high-level features, such as windowing functions and handling of keyboard and mouse input. The GLUT library is good for programs that are "simple to moderate" [6], in terms of user interaction i.e. simple key presses or mouse input. The GLUT library is not suitable for software that requires rich interaction, as it does not contain widgets i.e. buttons, radio buttons, checkboxes etc.

To provide a richer set of interaction features, toolkits such as GLUI and GLOW could be used. GLUI is a "GLUT based C++ user interface library which provides controls such as buttons, checkboxes, radio buttons and spinners to OpenGL applications" [7]. GLUI is window system independent and allows complex user interfaces to be built into OpenGL applications.

GLOW is "a cross-platform object-oriented framework for building interactive applications using OpenGL" [8]. The GLOW framework has classes containing all the code necessary for implementing a user interface. However the detail of the specific features of the interface must be provided by writing subclasses to the classes provided by GLOW.

2.3.3 Choice of Development Language

OpenGL seemed to provide enough functionality to develop the software required for this project. Also since it had been learnt before, the experience had been gained to be able to use it with ease. Java3D, on the other hand, would have required extensive knowledge of the Java language, which was not possessed. The effort that would have been required to learn Java and Java3D did not seem to be justified for a project of this scale when an equivalent alternative, OpenGL, was available.

Since the choice was made to use OpenGL, a choice had to be made as to which toolkit to use to develop the graphical user interface. In the analysis above, GLOW and GLUI were presented as possible options. GLOW is an advanced library, which contains many widgets and is also based around object-oriented concepts [8]. This means it is easily extensible. However while it contained many advanced features, many of the features were not needed for this project. This meant that if GLOW were used then a lot of unnecessary detail would have to be learnt. The GLUI library provided all the functionality required for this project and also seemed to be simpler than GLOW, in that it uses simple one or two line statements rather than an object oriented approach. This meant that the GLUI library was easier to learn than the GLOW library. Therefore the GLUI library was used.

2.4 Project Milestones

The following milestones were identified for the project. These are simply stages in the project and the completion of each stage indicated a milestone in the project: -

- Background research into security models and SQL Server 2000's security model
- Designing graphical metaphors to represent one security model
- Designing the software for one model (an enhancement is to produce designs and software for more than one model)
- Completing the implementation of the software for one model
- Test the software components that have been developed
- Evaluate the software and the project
- Write up the project report

To ensure the project ran smoothly and that the work was evenly spread, a project schedule was created. An attempt was made to ensure the schedule was realistic and the majority of the time the work remained on schedule. However there were times when the work got delayed. Therefore several versions of the project schedule were made as the project progressed. The stages relating to development of the software in these schedules were allocated according to the methodology chosen i.e. waterfall model with feedback. This means that the stages were iterative and that the design, implementation and evaluation of the software all overlapped in the schedule. All the schedules drawn up during the project are contained in Appendix B.

2.5 Summary

In this chapter, a choice has been made to use the 'waterfall model with feedback' methodology and a combination of OpenGL, GLUT and GLUI to develop the software. A project schedule has been drawn up and milestones have also been defined. In the next chapter, some of the major security models are discussed and their features are analysed.

Chapter 3 – Background Research

3.1 Introduction

In this chapter the basic concepts of computer security are introduced and a selection of important security models are investigated. These models are then compared to the model implemented in SQL Server 2000.

The security concepts discussed in this chapter are based on access control. Access control is about controlling access to an object or resource. Access control is necessary to ensure that only authorised people are allowed to view information or data. Some concepts that are frequently used in computer security literature will now be discussed.

Subjects, objects and operations are frequently mentioned in the discussion of access control. Subjects are usually users of a system, or in some cases are processes. Objects are usually resources containing data or information. Operations are the means a subject will use to access an object. Operations typically include read, write, execute and append.

These concepts have originated from some early work by Lampson [9]. Lampson described an "object system", consisting of a set of "objects", a set of "domains" and an "access control matrix". The 'objects' in this system are the same as described above. The 'domains' in Lampson's model refer to the subjects described above. The access control matrix in Lampson's model is a matrix with its rows labelled by subjects (domains) and its columns labelled by objects. Each element in the matrix contains a set of operations. An element *i,j* in the matrix defines the access operations allowed for subject *i* on object *j*. Lampson's work led to a model of access control where, "a subject *s*, is allowed access *a*, to an object *o*, only if *a* is included in the so^{th} element of the access control matrix"[9].

3.2 Security Models

A security policy refers to a policy used by a computer system to manage some form of access control. A security model concentrates on certain rules of a security policy and ignores irrelevant detail. A security model will usually be a formal representation of the security policy and will express the requirements of a policy mathematically. In this section, the main features of some of the major security models relevant to this project are described and a comparison of these models is presented.

Four security models are investigated in this chapter. These security models were chosen for differing reasons. The Bell-LaPadula (BLP) model was chosen as it is widely regarded as one of the earliest formal models and because many other models are based around the ideas it presents. The Harrison-Ruzzo-Ullman (HRU) model was chosen as it presents a very low-level security model, which has a

system-oriented approach. The Chinese Wall model was chosen as it presents a security model aimed at a specific enterprise and therefore presents a different notion of security. The Role Based Access Control (RBAC) model was chosen, as it is a user-oriented model, which seems to approach security from a user perspective rather than a system perspective. The RBAC model is also a more comprehensive model, which covers a lot more areas than the other models. This selection of security models was chosen as it presents a wide range of policies and each model has a different approach to security and a different of idea of what needs to be protected.

3.2.1 Bell-LaPadula Model

The Bell-LaPadula (BLP) model was one of the earliest formal models [10]. The model was developed primarily for government and military applications. The BLP model concentrates on classification of information and is based around making sure only authorised subjects can access a resource. Therefore it is sometimes referred to as a "confidentiality" model [11]. The aim of the model is to prevent declassification of information. This means it aims to prevent situations where confidential information will be downgraded to allow an unauthorised person to view it.

The model uses the concepts described previously in section 3.1. This means it has a set of subjects, a set of objects and an access control matrix. The access control matrix serves the same purpose as in Lampson's model, as described in section 3.1. The BLP model allows four types of access operations i.e. read, write, append (also known as blind-write) and execute.

The BLP model uses a concept that has not been described previously, this is a set of security levels that have a partial ordering of $\leq$. These security levels are used to classify each object in the domain. This means that every object has a security level associated with it. Furthermore, each subject has two levels associated with it. These levels are called the "clearance level" (or maximal level) and the "current level" [12]. The importance of these two levels will be clear when the rules of the model are discussed, but it should be noted that the current level cannot be higher than the clearance level. In this model the term "dominates" is used denote one level being greater than/equal to another level.

The BLP model defines security using states. Three security properties have been defined which determine whether a state is secure. These are the "ss-property", the "*-property" and the "ds-property" [10]. If the state of the system satisfies all three properties then the system is secure.

Simple Security Property (ss-property) – This property states "a subject can only read/write an object if the subject's clearance level dominates the objects classification level" [12]. This is described as a "no-read-up" policy [10]. As an example, we can imagine a subject S1, with a clearance level of 'Private'. Now imagine several objects distributed across three levels – Public,

Private and Secret. Where secret dominates private and private dominates public. Figure 3.1 shows the levels, which the subject S1 can read from. It is important to note that write access also includes the right to read an object (unless it is a blind write/append). This means that writing to an object would also be subject to the ss-property, as you would also be reading the object.

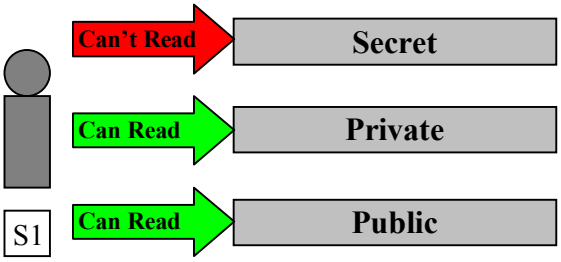


Figure 3.1: Diagram to Show ss-property

Star Property (*-property)

This property states "a subject can only write to an object if the classification of the object dominates the current level of the subject" [11]. This means that a subject cannot write to an object lower than it's current level. The way this affects the example used before is shown in figure 3.2 (assuming current level is same as clearance level). This diagram shows that S1 can write to files at the same level as itself i.e. at the private level. However it also shows that S1 can write to the Secret level. This is assumed to be a blind-write (append). This is allowed by the BLP model but is prohibited by most practical implementations.

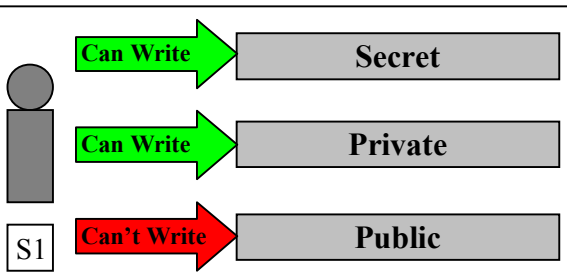


Figure 3.2: Diagram to Show *-property

The *-property also states that if a subject is writing to an object *o,* then all the other objects the subject can read/write must be dominated by *o*. This means that if S1 is writing to an object O1 at the Private level, then all the objects that S1 is reading must be dominated by O1. This means that all reads must be at either the private level or the public level and not at the secret level. This is shown in figure 3.3. Again note that a write access will also include the ability to read, therefore a when we say all reads must be either at the private level or the public level, this also applies to write accesses.

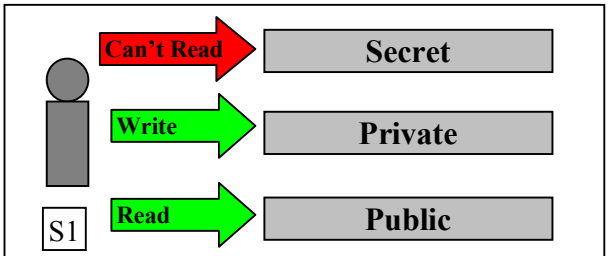


Figure 3.3: Combination of Reads & Writes

The *-property is a "no-write-down" policy [10]. This property is required in addition to the ss-property because the ss-property does not stop a low-level subject from creating a new file with a high level clearance which contains a program which could then read the high level objects and copy them to low level ones. This means that the ss-property allows some declassification of information. The *-

property prevents this declassification, but imposes the restriction that a subject cannot write to any object at a lower level. The authors of the model have identified two ways to get around this situation. Firstly, you will notice that the *-property uses the subjects 'current' level rather than the clearance level. This is used as it allows us to lower the current level, thereby allowing the subject to write to lower levels, while still maintaining the same clearance level. Another interpretation of this is that the current level indicates the level at which the subject has logged on. So a subject could change their current level by logging off and logging on again at a different level. The second method of getting around this restriction is to allow some subjects to violate the *-property. These subjects are known as "trusted" subjects [11]. Some authors, such as [12], mention that the *-property is meant for processes rather than humans and therefore, human subjects should be allowed to violate the *-property. This seems to make sense as human users can remember or write down details from files and therefore could declassify information without violating the *-property anyway.

Discretionary Security Property (ds-property)

The ds-property describes the discretionary policy (i.e. at the discretion of a subject). This property states "each access made must be granted to that subject for that object in the access control matrix" [12] i.e. the subject must be allowed, according to the access control matrix, to access the object. This is discretionary because it relates to a user definable access control matrix i.e. the subject that controls/owns the file can give other subjects access to an object through the access control matrix.

Basic Security Theorem

A state is secure if the ss, star and ds properties are all satisfied. A state transition is secure if it goes from one secure state to another secure state. This leads to the basic security theorem, which states, "if all state transitions in a system and the initial state are secure, then every subsequent state is also secure" [10]. This means that if you can prove that the old state and the new state are secure, and that this is true for every state transition then the system is secure

3.2.2 Harrison-Ruzzo-Ullman Model (HRU)

The BLP model does not state policies for changing access rights or for the creation or deletion of objects. The Harrison-Ruzzo-Ullman model (HRU) devised in 1976 [13] captures the notion of changing access rights and the addition and removal of subjects and objects. The HRU model uses a set *S* of subjects, a set *O* of objects and a set *R* of access rights. The access rights in this model are slightly more extensive than those of the BLP model. They are likely to contain the rights to 'own', 'read', 'write', 'add', 'execute' and 'control' objects. The central concept of the model is an access matrix, represented by *M*. It is important to note that the HRU model classes every subject as also

being an object. Thus a subject can carry out operations on other subjects (i.e. control their access rights using the 'control' right).

The HRU model is based around an access control matrix, which contains the subjects as rows, the subjects & objects as columns, (as subjects are also classed as objects), and the access rights available to a subject on an object as the elements of the matrix. This is shown in figure 3.4.

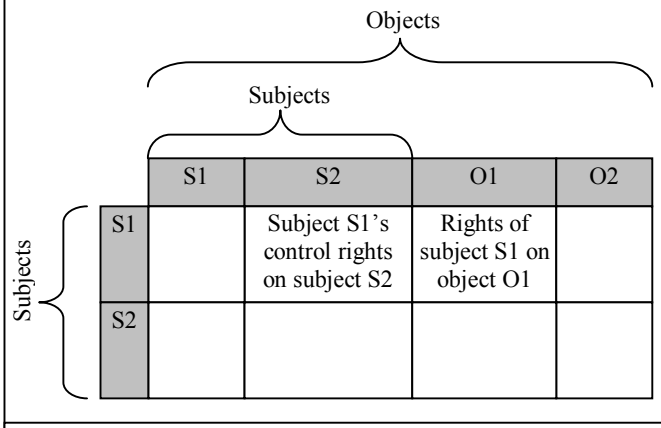


Figure 3.4: Access Control Matrix for HRU Model

The model allows subjects to modify the access control matrix through commands. These commands will differ depending on the situation the model is used in, but they will all have the same structure. This structure involves conditions and primitive operations. Conditions will query the state of the access control matrix i.e. is a certain right specified for this subject on this object? Or does a certain subject exist in the matrix? etc. The primitive operations in the command will make changes to the access control matrix. A single command may include more than one primitive operation. There are six primitive operations, which have been defined for the HRU model [14]. These are: -

- enter right *r* into M_{SO}
- delete right *r* from M_{SO}
- create subject *s*
- delete subject *s*
- create object *o*
- delete object *o*

These commands are recorded as changes to the access matrix. Therefore the access matrix describes the state of the system and the above commands are the state transitions of the model. These operations are shown in a visual form in figures 3.5-3.8 (see right). The figures represent the access control matrix (grey block) and how the operations alter the matrix. In these figures s, s1 and s2 are subjects and o, o1 and o2 are objects.

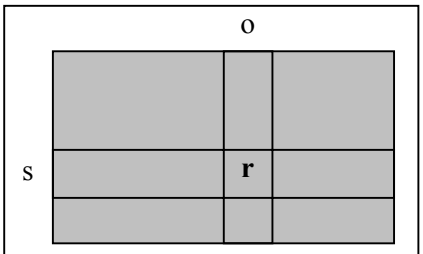


Figure 3.5: enter right r into M_{SO}

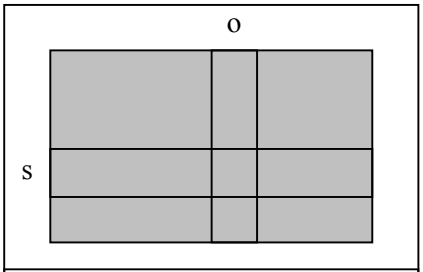


Figure 3.6: delete right r from M_{SO}

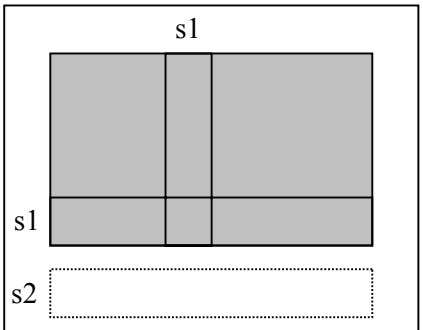


Figure 3.7: create subject *s1* delete subject *s2*

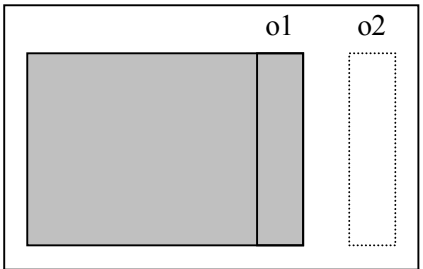


Figure 3.8: create object *o1* delete object *o2*

A security policy in HRU can regulate the allocation of access rights. To check that a system complies with a policy you must check that the system does not allow illegal access rights to be granted through the use of a legal command. This means that there should be no commands that are legal according to the security model, but which allow a subject to be granted access to an object which they should not be allowed to access. The two properties that describe this situation are "leak" and "safe".

> "An access matrix *M* is said to **leak** the right *r* if there exists a command *c* that adds the right *r* to a position in the access matrix that previously did not contain *r*." **Source [11]**

This rule relates to situations when a subject is about to carry out an operation (usually giving another subject a right). The subject should be able to determine whether or not the operation will lead to an unauthorised subject getting access to an object. Obviously, there will be subjects who are expected to have the right, these are known as "reliable" subjects [14]. These subjects are "trusted" and the authors of the model suggest they should not be included when the leak property is evaluated. If, after removal of reliable subjects, the access matrix leaks the right then the state is "unsafe". However if the right is not leaked then the state is said to be "safe" [14].

> "An access matrix *M* is said to be **safe** with respect to the right *r* if there are no sequence of commands that can transform *M* into a state that leaks *r*." **Source [11]**

The authors of the model have proved that the "safe" property of an arbitrary access control matrix with respect to an arbitrary access right is "**undecidable"** [13] i.e. you cannot verify that the matrix is safe. However the authors of the model have proved that the "safe" property is **decidable,** if commands contain only a single primitive operation [13]. This means that if the system is not very complex, then it is easy to verify the security of the system. But with multi-operation commands the complexity is too large, hence we cannot decide whether the eventual state reached is safe or not. This is a considerable limitation of this model, as the majority of situations will require cases where commands must include multiple operations.

3.2.3 Chinese-Wall Model

The Chinese Wall Model was proposed by Brewer and Nash [15]. This model relates to consultancy businesses where analysts deal with many clients. These clients may be competing with each other and the analyst must make sure no "conflicts of interest" arise. The basic rule of this model is: "There must be no information flow which causes a conflict of interest" [12].

The Chinese Wall Model has a set of objects, a set of companies, a set of subjects and a set of conflict of interest classes. The basic concept of this model is that all objects are data sources that relate to one company in the company set. The model takes all objects that concern the same company and groups

them into a dataset. Furthermore, some of the companies in the system may be in competition with each other. Therefore, all the datasets are then grouped into conflict of interest classes, such that each class contains datasets of companies that are in competition with each other. This is shown in figure 3.9 below. There may also be certain objects that have been cleaned in a way that all sensitive information has been removed. These objects are known as sanitised objects and can be accessed without any restrictions being applied to them. This model maintains a 'history' matrix to record whether a subject has ever had access to an object. This is because conflicts of interest can arise from previous accesses as well as current accesses.

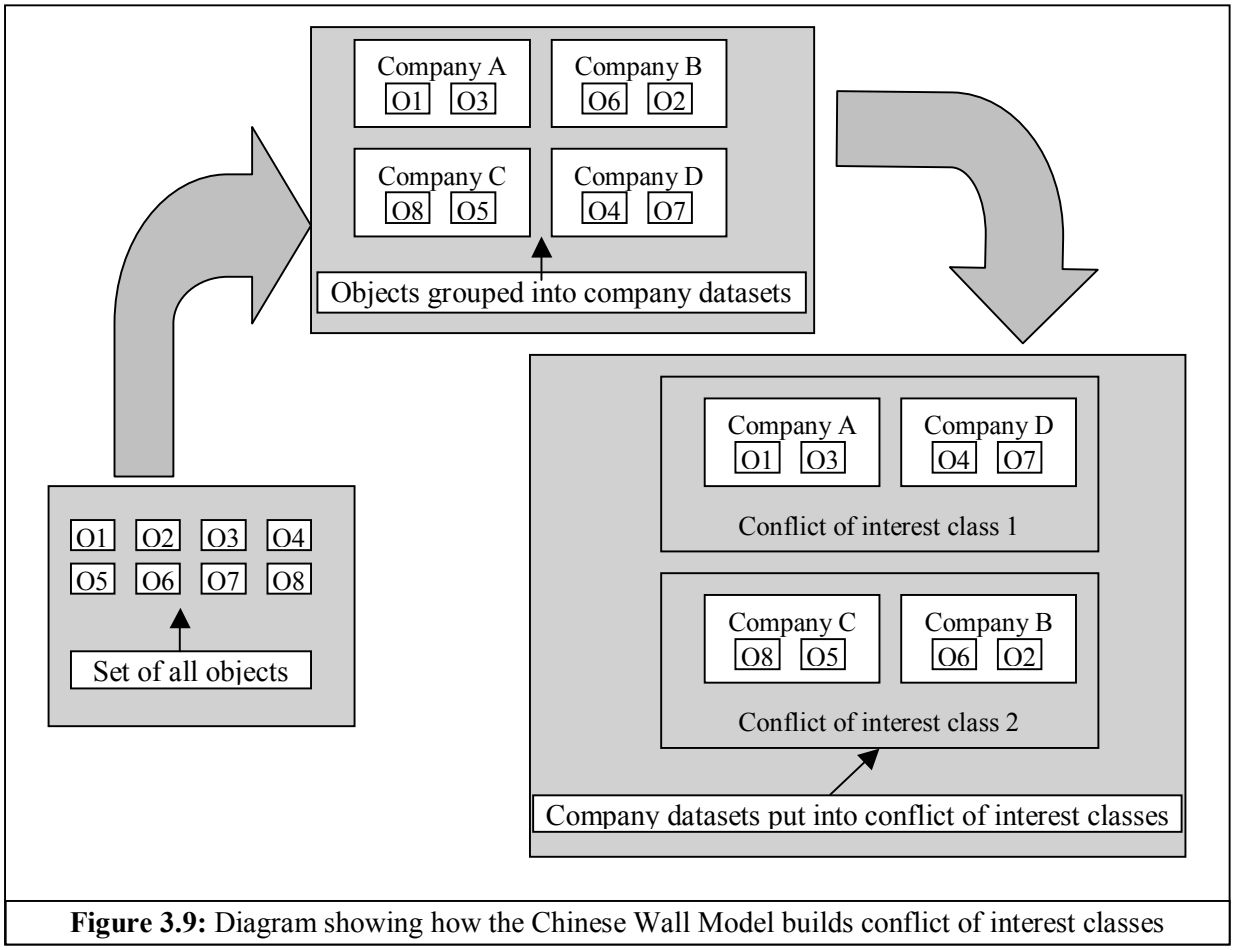


Figure 3.9: Diagram showing how the Chinese Wall Model builds conflict of interest classes

The model has two properties, the ss-property and the *-property. The ss-property states "a subject can only access an object if the subject already has access to another object from the same company (dataset) or that the subject does not hold an object, which would cause a conflict of interest with the new object" [15]. This means that a subject can access at most one company dataset from each conflict of interest class. For this rule, the history matrix is used to check whether a subject has ever had access to an object. This matrix contains a record of all the objects a user has accessed in the past. So if the subject has read an object and released it, then according to the model, the user may still remember about it and therefore will be denied access to any dataset that belongs to the same conflict of interest class.

This property alone does not stop indirect information flow. A subject could still put sensitive information into an intermediate object, which could then be accessed. This is similar to the idea that was discussed for the BLP model which led to the *-property in the BLP model. Similarly, a *-property is required here too. The *-property states that "a subject **cannot** write to an object if the subject can read any object, which is in a different company dataset and contains unsanitised information" [15]. Furthermore to be able to write to an object the ss-property must also be satisfied. This means that if a subject is reading some sensitive (unsanitised) information from company 'A' then the subject will never be able to write to an object belonging to another company. This is because this model uses a history matrix and therefore the read access on dataset 'A' will be recorded as true.

This is an important point as it shows the restrictions the model places. This restriction would not be realistic in most situations and therefore it may be the case that the *-property is designed for processes rather than human users, and in a similar way to the BLP model, there may be "trusted" subjects who may be able to violate the *-property. Although it has to be emphasised, the authors of the model do not make any reference to this limitation and therefore it has to be assumed that this property applies to all human users. Also this model does not have static access rights. This is because as a subject accesses objects, the subject's access rights are gradually decreasing. Therefore access rights have to be checked in every state transition.

A problem has been pointed out with this model in [12]. It is shown that the partitioning of datasets into conflict of interest classes assumes that conflict of interest is "transitive" [12]. This means that if A has a conflict of interest with B and B has a conflict of interest with C, then A, B and C will all be in a single conflict of interest class, even though A may not be in competition with C. This imposes restrictions which would be unrealistic in the real world i.e. a user who accessed A would be denied access to C in the example above, even though the access is not likely to cause any problems.

3.2.4 Role Based Access Control Model (RBAC)

There are many versions of the role-based access control model that have been proposed. In this section one of these models, proposed by Ferraiolo et al. [16], is investigated. In an RBAC model access decisions are based on roles individual users have as part of an organisation. The RBAC model in [16] contains users/subjects, roles, operations and objects. "A user/subject is a person, a role is a collection of job functions and an operation represents a particular mode of access to a set of one or more protected RBAC objects" [16]. A user may be associated with many roles and the operations associated with a role constrain the user's actions. The RBAC model in [16] is essentially based on nine rules. These rules are constraints, which control the activities of subjects. The following subsections will describe each rule.

Rule 1 – Role Hierarchy

This rule states, "if a subject is allowed to access a role and that role contains another role, then the subject is also allowed to access the contained role" [16]. This rule creates a hierarchy of roles, where some roles contain other roles to form a tree-like structure. This is provided, as there may be situations where many roles in an organisation will need to perform the same operations. A role hierarchy simplifies management of such situations by assigning the common operation to one role and then allowing the other roles to contain this role. If the concept of a role hierarchy were not provided then this operation would have to be authorised for each role separately. For example imagine three roles – employee, accountant and manager. The employee role has some operations authorised for it. The accountant and manager roles contain all the operations of the employee role plus some other operations. If role hierarchies aren't provided then the operations must be defined for all three roles separately. However with a role hierarchy you could simply authorise the operations for the employee role and allow the accountant and manager roles to contain the employee role. Figure 3.10 shows this role hierarchy.

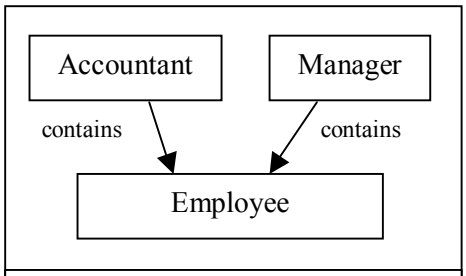


Figure 3.10: Example Role Hierarchy

Rule 2 – Static Separation of Duty

This rule states, "A user is authorized as a member of a role only if that role is not mutually exclusive with any of the other roles for which the user already possesses membership" [16]. This rule deals with mutually exclusive roles and is a way of separating roles so that one person cannot be a member of two mutually exclusive roles. For example, it could be the case that the manager and the accountant in an organisation cannot be the same person. Then these two roles could be described as mutually exclusive and any user would only be allowed to be a member of at most one of these roles, not both. This rule relates to the stage when subjects are assigned to be members of roles i.e. when the administrator assigns the role to the user. This is why it is described as static. This is different to dynamic separation of duty, which is covered later.

Rule 3 – Cardinality

This rule says that there may be some upper limit for a role. It states, "The capacity of a role cannot be exceeded by an additional role member" [16]. The capacity of a role could apply at the stage when roles are assigned to subjects or at the stage when a subject actually becomes active in a role. For this project the latter is assumed. Therefore there may be a limit to the number of users that may be active in a role at any one time. For example, in certain organisations it may be important to ensure that there is only ever one user active in the manager role.

Rule 4 – Role Authorization

This rule states, "A subject can never have an active role that is not authorised for that subject" [16]. This rule means that a subject can only become activate in roles that have been authorised for them. So if a subject is authorised to be a member of the employee and accountant roles, then the subject can become active only in these roles and not in any other roles that may exist. This rule may seem very simple, but it is required as a sub rule for more complex rules in the model.

Rule 5 – Role Execution

This rule states, "A subject can execute an operation only if the subject is acting within an active role" [16]. This rule is about carrying out operations and it means that the subject must activate a role before they can execute any operations. This rule simply says that to execute an operation, the subject must be active in **any** role. It does not actually specify that the role the user is active in, must allow the operation. Again this is a simple rule, but is used later, in rules 7 & 9 to form more complex rules.

Rule 6 – Dynamic Separation of Duty

This rule states, "A subject can become active in a new role only if the proposed role is not mutually exclusive with any of the roles in which the subject is currently active" [16]. This means a subject cannot activate a role, if the role is mutually exclusive with another role, which the subject has already activated. This is different to rule 2, which relates to static separation of duty. This is because rule 2, deals with separation of duty at the time a subject is made a member of a role, whereas this rule deals with separation of duty when a role is activated. So back to the example used for rule 2, the user this time could be authorised as a member of both the manager and the accountant role. However, this rule prohibits a user from being **active** in both these roles simultaneously if they are defined as mutually exclusive. This is different to static separation of duty where a user would only be allowed **membership** of at most one of these roles. For dynamic separation of duty, a user would be allowed to be a **member of both roles** but would only be allowed to be **active** in at most one role at once.

Rule 7 – Operation Authorization

This rule states, "a subject can execute an operation only if the operation is authorised for the role in which the subject is currently active" [16]. This rule is a combination of rules 4 and 5 and adds the constraint that the operation the user requests is actually authorised for the role the user is active in. So referring back to rule 4, a subject will not be able to activate a role unless they are authorised to do so. Furthermore, rule 5 says that a subject will not be able to execute an operation until they activate a role. This rule adds that the active role must have the operation authorised for it in order for the operation to be executed.

Rule 8 – Operational Separation of Duty

This rule is provided for situations where there is an overall business function that is carried out through a series of smaller functions. This overall function will be such that a single person should not be allowed to carry out the entire function. Thus it must be ensured that there exists no role that allows the entire business function to be completed. For example, there may be a business function for the purchase of new stock. This may involve, placing an internal order, getting financial clearance and placing the actual order for the stock. It may be a business requirement to ensure that a single person cannot carry out this whole process. Therefore, the security model must ensure that no single role is allowed to carry out all three functions.

This rule states, "A role can be associated with an operation of a business function only if the role is an authorized role for the subject and the role had not been assigned previously to all of the other operations " [16]. This rule says that for all the operations associated with a business function, no single role can be allowed to perform all of these operations. This rule relates to business functions rather than operations, therefore the model provides a means of defining business functions through the operations they require. It should be noted that while a single role would not be allowed to perform all the operations for a business function, it may still be possible for a user to be made a member of two or more roles which together allowed the subject to perform all the operations for a business function. This issue could possibly be resolved by defining mutually exclusive roles.

Rule 9 – Object Access Authorization

This rule is a combination of some of the other rules specified previously. This says that a subject can carry out an operation on an object only "… if the role is part of the subject's active role set …" (Rule 5), "… the role is allowed to perform the operation …" (Rule 7), and "… the operation to access the object is authorized." [16]. So this rule adds the extra constraint that the operation being requested is actually performable on the object (i.e. it may not be possible to perform certain operations on an object i.e. executing a text file). Extending from rule 7, a subject must be authorised for the role, the subject must be active in this role, this role must have the operation authorised for it and the operation must be performable on the object in order for the operation to be executed.

Other RBAC Models

As was mentioned earlier, the Ferraiolo et al. model is a specific version of the RBAC model. Other RBAC models have also been proposed. There are slight differences between the different RBAC models, but most have the same core principles. Sandhu et al [17], have proposed a similar RBAC model. In their model, they propose a hierarchy of RBAC models. They propose four types of model $RBAC_0$, $RBAC_1$, $RBAC_2$ and $RBAC_3$. $RBAC_0$ is regarded as the base model. This contains the core concepts of users, roles and permissions. This is regarded as the least needed to provide role based

access control. From $RBAC_0$ you can go up to $RBAC_1$ or $RBAC_2$, both of these contain $RBAC_0$. $RBAC_1$ provides the concept of role hierarchies while $RBAC_2$ provides constraints such as separation of duty, cardinality etc. From either of these two you can go up to the next level, $RBAC_3$. This combines all the models discussed so far i.e. provides both role hierarchies and constraints. The difference between Sandhu et al's model and Ferraiolo et al's model is that the latter combines everything into one model whereas the former breaks the model down into separate components. Ferraiolo et al's RBAC model would be classified as $RBAC_3$ according to Sandhu et al's model as it provides role hierarchies and constraints all in a single model.

Conclusion on RBAC

The principal motivations behind RBAC, according to Ferraiolo et al., are "the ability to express and enforce enterprise-specific security policies and streamline the typically burdensome process of security management" [16]. This is an aim, which it seems to achieve. The RBAC model is adaptable to a large number of organisational structures. It covers a wide range of policies relating to static and dynamic access policies. The model is a great help to security administrators as it gives them much more flexibility in the way they work. It also allows security administrators to work at a level of abstraction, through roles, that is easy to use and understand, rather than working at the lower level of granting and denying access permissions through access control matrices.

3.2.5 Comparison Of The Security Models

In this section, the four models discussed previously are compared to each other in an attempt to highlight the similarities and the differences between the models and the way they approach security. Firstly, the differences in the way each model approaches security are discussed, and then the models are compared to each other one by one.

The first thing to mention is that all four models have different aims. The Bell-LaPadula (BLP) model aims to prevent declassification of information. Whereas the Harrison-Ruzzo-Ullman (HRU) model aims to capture changes in access rights and aims to prevent access rights leaking to unauthorised users. The Chinese Wall (CW) model is aimed at a much narrower field and aims to prevent conflicts of interest arising and aims to prevent fraud. The Role Based Access Control (RBAC) model covers many more areas than the other model but it aims to make system administration more manageable and aims to provide policies, which would suit many enterprises. So while all the models are aiming to prevent some form of unauthorised access, they all go about it in different ways.

Now the models are compared to one another. First, the BLP model is compared to the HRU model. Both these models are very system oriented and seem to concentrate on the protection of files and fit in the human needs around this protection. Also, both are based on access control matrices. However,

whilst the HRU model is based solely around the access control matrix, the BLP model uses the access control matrix as a secondary concept and goes beyond the matrix. The BLP model focuses more on the way objects are accessed rather than the actual state of the access control matrix as the HRU model does. Also, the properties of the BLP model and the HRU model relate to different things. While the HRU model concentrates on making sure modifications to the access control matrix are secure, the BLP model's properties aim to ensure that the accessing of objects does not violate the security policy. However, it could be argued that the ds-property of the BLP model is similar to the HRU model's properties as it relates directly to changes made to the access control matrix.

Next the BLP and CW models are compared. These models have a different approach to what security is. As mentioned before, the BLP model is very system oriented. The CW model seems less system oriented as it takes into account some user specifics but is still concentrated around protecting data. Both these models use an access control matrix, but the BLP model uses layers as it's main concept whereas the CW model uses conflict of interest classes. Also both models use the access control matrix for different reasons. The BLP model uses it to record the access rights of each user whereas the CW model uses it to record any accesses that users have made in the past. The properties of the two models are similar in that they both aim to stop subjects accessing resources they are not allowed to and also, they both aim to stop sensitive information being transferred between objects. However, the models have a different view of what to protect. The BLP model aims to stop information being declassified whereas the CW model aims to stop information from competing sources being accessed.

In this paragraph the BLP and RBAC models are compared. The RBAC model is much more general than the BLP model as the BLP model relates to certain enterprises whereas RBAC can be adapted to suit many situations. The RBAC model assigns subjects to roles whereas the BLP model assigns subjects to levels. Also in RBAC, users rights are controlled by manipulating roles, whereas in the BLP model, you can control users rights by changing their clearance level or by manipulating the access control matrix. The BLP method of changing access rights is less flexible than roles as changing the clearance level allows access to a lot more new objects and if you manipulate the access control matrix then you must ensure that you don't authorise access to objects above the subject's clearance level. But it could also be argued that changing the rights of a role may affect more subjects.

Now the HRU and the CW models are compared. This comparison is actually quite similar to the comparison made between the BLP and HRU models above. This is because both models use an access control matrix but the CW model extends upon the access control matrix through conflict of interest classes. Also the CW model uses a "history matrix" rather than an access control matrix, which means that every access is recorded and remembered. The CW model has a different notion of

security than the HRU model. While the HRU model regulates changes to the access control matrix, the CW model prevents fraud by ensuring subjects don't access competing companies resources.

Now the HRU and RBAC models are compared. The HRU model grants rights through an access control matrix, whereas the RBAC model grants rights to roles and makes subjects members of these roles. The HRU model is a primitive model when compared to RBAC as the HRU model tackles security from a system viewpoint whereas the RBAC model tackles security from a more human viewpoint. While both models regulate the allocation of access rights, they do it in different ways.

Finally, the CW and RBAC models are compared. The CW model assigns objects to conflict of interest classes, therefore we can say that it uses an indirection of conflict of interest classes between subjects and objects. Whereas, the RBAC model uses roles as an indirection between subjects and objects. Furthermore, the RBAC model allows you to modify subject's rights through roles. But the CW model doesn't allow modification of rights, as rights can never increase. Rights will only decrease as the subject makes accesses. Therefore the CW model can be regarded as being more restrictive than the RBAC model. Also the RBAC model isn't targeted at a specific situation, therefore its access policies are different to the CW model's policies and are more generalised.

3.3 SQL Server 2000's Security Model

In this section the way security is modelled in SQL Server 2000 is investigated and is then compared to the security models discussed previously. An overview of the features of SQL Server 2000's security model is also given in this section. Much of the information here is adapted from [18]&[19].

3.3.1 Analysis of SQL Server 2000's Security Model

Roles – Roles allow a user to be grouped into a single unit to which permissions can be applied. Permissions are granted to roles and users are made members of a role. SQL Server allows the nesting of roles and allows users to be members of many roles simultaneously. There are several different types of role in SQL Server 2000. These include the "Public Role", which provides default permissions for users in a database, "predefined roles", which have implied permissions and can be either, fixed server roles (administrative) or fixed database roles. Other types of roles in SQL Server 2000 are the user defined roles and the application roles.

Access Control – In SQL Server 2000, permissions are granted either directly to a user or to a role, which may contain many users. There are two types of permission that can be granted, statement permissions and object permissions. Statement permissions restrict who can create a database or items in a database e.g. CREATE DATABASE, CREATE TABLE etc. Object permissions restrict access to objects like tables, views, stored procedures etc. i.e. select, insert etc.

Permissions System – SQL Server uses an additive model for permissions i.e. the user gets the sum of the respective permissions of each role. But restrictive permission, such as DENY, takes precedence. Therefore if a user has read and deny permission on a single object then he will be denied access. There are 3 ways of controlling permissions in SQL Server, GRANT, DENY and REVOKE.

Ownership Chains – In SQL Server there is the concept of a "broken ownership chain". Ownership chains are established when permissions on an object are checked. A broken ownership chain occurs when an object does not have the same owner as the underlying object upon which it is based. SQL Server checks the permission of underlying objects when there is a broken ownership chain. This concept guards against a user gaining unwanted access to data.

3.3.2 Comparison To Other Security Models

SQL Server's security model has a big emphasis on roles and is therefore quite similar to the Role Based Access Control (RBAC) model discussed in 3.2.4. SQL Server uses the concepts of role authorisation and role activation. It also uses the concept of a role hierarchy, which it calls role nesting. However SQL Server does not implement all of the features described in the RBAC section. Also because it uses roles, it implicitly uses some of the concepts of the Bell-LaPadula (BLP) model (section 3.2.1), in that it grants users access, depending on their role and what it allows them to do, which is similar to the security levels for subjects in BLP.

There are also similarities between this model and the Harrison-Ruzzo-Ullman (HRU) model (section 3.2.2) in the way permissions are granted. The GRANT and REVOKE permissions are the same as the concept of entering and removing permissions from the access matrix in the HRU model. However SQL Server takes it a step further by also allowing access to be denied rather than revoked.

SQL Server's model does not include much about separation of duty, as it is really more of a specific business issue. This means that this model is not really comparable to the Chinese Wall model (section 3.2.3) or some of the elements of the RBAC model. However SQL Server uses the concept of ownership chains, which is a fairly powerful concept, but this concept is not covered in any of the security models discussed previously.

3.4 Summary

In this chapter, some formal security models have been discussed and their main concepts have been explained. These models have also been compared to each other. Furthermore, the security model of SQL Server 2000 has also been analysed and compared to the formal models discussed earlier. In the next chapter, some of the formal models discussed in this chapter will be selected and some metaphors will be designed in order to represent these models.

Chapter 4 – Devising Graphical Metaphors

4.1 Introduction

In this chapter, the security models discussed previously are mapped onto graphical metaphors. Due to the time limitations and the limited scope of this project, it was not possible to devise graphical metaphors for all of the models discussed previously. Therefore, only the Bell-LaPadula (BLP) model and the Role Based Access Control (RBAC) model were developed any further.

These models were chosen as they have very different approaches to security. The BLP model deals with security from a system perspective whereas the RBAC model takes a more human perspective. This contrast made it interesting to see how these differences affected the metaphors developed. The HRU model wasn't chosen, as it doesn't have enough depth to justify it being developed as a metaphor. The Chinese Wall model wasn't chosen as it is a very specific model and can only be applied to specific situations, thus the metaphors might only be applicable to specific environments.

4.2 What Makes A Good Graphical Metaphor

When developing graphical metaphors, it is necessary to understand what elements make a good metaphor. A metaphor is a way of reducing the complexity of a concept by representing it as some similar object about which the user has some knowledge. Hosmer presents some interesting ideas about visual icons in [20], much of this work is also applicable to graphical metaphors.

A metaphor should be "easily understandable" to the user [20]. Upon seeing or using the graphical metaphor the user should have some understanding of what it depicts. A good graphical metaphor will be "intuitive" and "easy to understand and use"[20]. Furthermore a good metaphor would also be reusable in different contexts i.e. to represent different scenarios.

4.3 What Methodology Should Be Used To Develop A Graphical Metaphor

Generating graphical metaphors is an unusual concept in that there aren't any standard methodologies designed for the purpose. There are no standard guidelines, which take you through developing metaphors. An article by Wilson, Rauch and Paige [21] attempts to develop an approach to designing graphical metaphors. In this article the authors claim that designers need to draw on a "disparate set of skills" and experiences, in a variety of fields of knowledge, to develop suitable metaphors. They believe that to design a good metaphor you need to begin by carrying out some "task analysis" [21]. By this they mean that the designers should develop "a thorough understanding of the problem domain". In the case of this project this would mean having a detailed understanding of exactly how the security model works and what the major concepts are.

After this the methodology described in [21], says designers should develop the metaphor by studying the users "mental model". They believe that users have a mental model that "filters anything and everything we give them in an application". The authors of [21] believe that by studying users mental models, designers can utilise these mental models to increase understanding of the underlying concepts. In the case of this project, this relates to linking the security model to concepts the user has knowledge about i.e. the metaphor. But what this approach is telling us is that the metaphor we use should represent the security model in a way that matches the users mental model. Otherwise the user may not understand the security model properly. This means that the user may take the metaphor more seriously than expected and have false understandings of what the security model actually does.

For this part of the project, the methodology described above was used. However this methodology was altered to suit the needs of this project. The first step in the methodology is to identify the major concepts of the security models through "task analysis". This has already been done in chapter 3. However the major concepts are revisited again. The next step is to use these concepts to determine a suitable metaphor, taking into account the possible aspects of the users mental model and how these might affect the use of the metaphor. This part of the methodology was altered so that less use was made of the users "mental model". This is because it was more relevant for this project to evaluate whether the metaphor that was eventually used actually fitted in with the users mental model, rather than trying to design a metaphor around a specific set of users mental models. This was due to the fact that it was not clear exactly what the software would be used for and therefore it was not suitable to design the metaphors with a specific set of users in mind.

As mentioned above, it was not possible to identify the exactly who **will** use the resulting software or for what purpose. However it was possible to identify **'**potential' users who **may** use the resulting software. The current software could potentially be used in three main areas:

1. To monitor the usage of a system i.e. monitor who is accessing files, what access rights have been granted and to possibly find security flaws in the system.
2. Another possible use would be to use the software to grant and deny access rights to users. This could be for administrators or even for users to view and change their own access rights
3. The software could also be used as an educational tool. This would be for someone who wants to learn about security models. It may help students and others to learn more about the security models by actually interacting with the software.

While these are potential uses for this type of software, the software was originally intended for the first usage above i.e. to allow the monitoring of system usage. The potential users of such software would have been system/network administrators, as they would be the ones that would need to monitor usage. However, as the project progressed, it became clear that the software may not be as

useful to system administrators as first imagined. This is because the software may not highlight security flaws in the system as easily as was first thought. The software could have been developed for the second usage above. However, such software would probably be more useful in realistic situations where the software was manipulating real access rights rather than in a simulated scenario. If a simulated scenario is used then the software goes more into the realm of the third usage described above. Ultimately the software was developed for people wanting to use it to learn about security models. Students and others learning about security models will find it useful, as the software more of a simulation type of software, rather than software that can be used for proper access monitoring or access rights management. The software allows users to simulate different situations and allows them to see what events occur in these situations. The software was developed for this purpose, so potential users are likely to be students or others wanting to learn about security models. As mentioned before, the mental model was not taken into account very much when designing the metaphors, but this analysis of usages was helpful when evaluating the software as it allowed specific users to be targeted.

4.4 Mapping The Security Models Onto Graphical Metaphors

In this section the security models are actually be mapped onto graphical metaphors. The graphical metaphors are developed with a specific environment in mind. For example, the BLP model metaphor has been designed for a context in which there are a number of shared files. Therefore operations such as read, write and execute are used. Whereas for the RBAC model, a database environment was used. This means tables, and operations such as select, insert, delete etc. This approach was chosen as it allowed a broader understanding of the way metaphors could be applied to different scenarios. This is because there are differences in the way these two environments operate.

4.4.1 Bell-LaPadula (BLP) Model

4.4.1.1 Main Concepts Revisited

The BLP model was designed to model confidentiality. The aim of the model was to prevent the flow of information from a high to lower level. It uses an access control matrix and the concept of security levels. This model is described in more detail in the section 3.2.1. Analysis of the model done in section 3.2.1 showed the main concepts used in the model to be: -

- Subjects i.e. users
- Objects i.e. files
- Accesses/Operations i.e. read, write and execute.
- Security Levels (partially ordered ≤)

These aspects were therefore represented in the graphical metaphor as they represent the domain of the model. However, from this the major concept of the model needed to be identified i.e. the idea

upon which the model is based. The major concept is actually the notion of security levels. These levels are the basis of this model as these are the properties that determine whether or not access is granted. It should be noted that while this model is based around the idea of security levels, it also uses an access control matrix. The purpose of this matrix is to define the actual access operations granted to each subject on each object. So for example, the clearance (maximal) level of the subject may allow him/her to access an object, but the access control matrix may only allow this subject to read the object and not write to or execute the object.

Also, these security levels have a partial ordering of ≤. This is important, as the metaphor also needed to represent this partial ordering. Also it should be noted that subjects are being considered as users, therefore the temporary downgrading of subjects using their current level was not considered to be applicable as this is the approach used for processes, rather than human users of a system. Processes were not considered, as they are not likely to work with graphical metaphors. Instead of having this system of downgrading subjects, it was assumed that subjects could write to files below their clearance level. This is similar to the concept of trusted subjects described in the section 3.2.1. However they were not allowed to perform a write down (see *-property in section 3.2.1). So while they were trusted subjects in one sense, they were still not allowed to declassify information. But, an option was to be provided to allow users to treat the subjects in the metaphor as processes. Activating this option would mean that subjects wouldn't be allowed to write to files below their clearance level.

4.4.1.2 Designing The Graphical Metaphor

After analysing the model and identifying the main concepts, it was time to devise a metaphor onto which these concepts could be mapped. The first stage was to brainstorm some ideas that may fit the concepts described above. After devising some initial ideas, the two metaphors that seemed most reasonable were concentric rings and a layered representation.

Concentric Rings

The concentric rings metaphor would model the set of security levels as a set of concentric rings. With the lowest level being the outermost ring and the levels getting higher as we moved inside the ring. This is something that may be linked to the users mental model, as it seems logical to assume that to get to the central ring you would need permission to access the outer rings that come before it.

Objects could have been visualised as residing within the ring that corresponded to that objects security level. Furthermore, subjects could have been represented as moving through the model as they carried out access operations. This seemed like a good idea at first, but upon closer inspection, it was discovered that this model would have problems representing a partial ordering of ≤. This was because the rings would be most suited to a linear ordering of levels, where one dominates the other, rather than a partial ordering, where there may be elements that are not comparable. This partial

ordering could have been represented, by simply dividing the corresponding ring into two or more semi rings. However this may have been a problem as users may misunderstand the concept and may interpret this split as meaning that the two levels are somehow related as they are still in the same ring even though they are split. An example is shown in Figure 4.1 below.

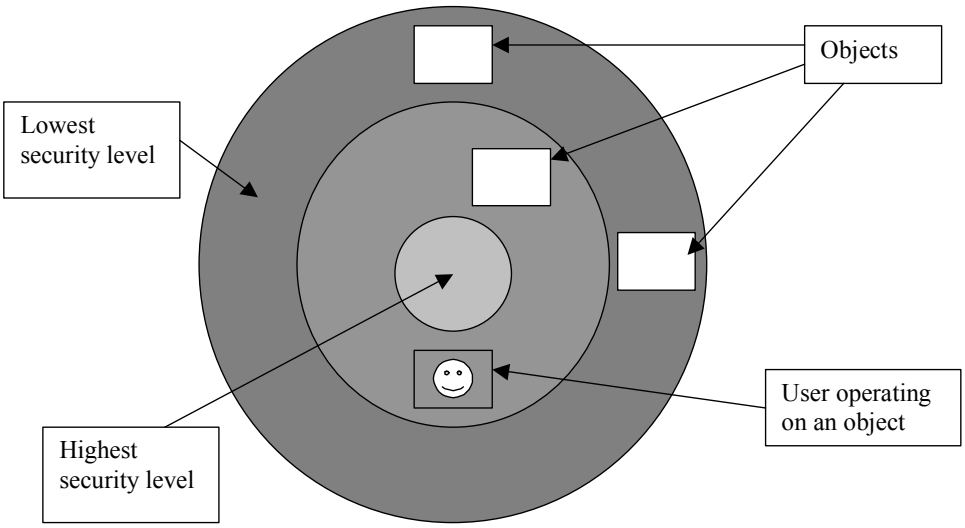


Figure 4.1: Concentric rings metaphor

Layered Representation

The layered architecture metaphor is one that is commonly used in computing to describe protocols. Using a similar concept we could represent the BLP model using a set of layers. Each layer in the metaphor would model a specific security level. With this metaphor the lowest layer would represent the lowest security level.

As for the concentric rings, objects would be placed in the layer with the corresponding security level. Subjects carrying out access operations would be represented by icons showing the nature of the access. These icons would be placed inside the actual object being accessed.

This model would also allow us to represent a partial ordering $\leq$, of security levels. As we could represent layers at the same level that are not connected, indicating that they are not equal but also that one layer is not higher than the other. However, to do this properly the layers would need to be broken up into a tree like structure. Because if the layers remained adjoined as in a regular layered architecture, the user may believe they are somehow related or maybe the same level. Therefore they were split up to form a sort of tree like layered structure. Figure 4.2 below shows how this would be represented. This metaphor was the better of the two and was developed further into graphical software.

Figure 4.2 below shows how read and write operations would be represented through this metaphor. This metaphor distinguished between objects that are accessible to the user and ones that are not. Also the metaphor represented the access control matrix by representing operations granted to the user differently to ones that are not granted to the user. If a layer, object or operation was not authorised for a subject then it was represented with a different colour (i.e. greyed out). Furthermore, the metaphor also indicated the subject's clearance level. It should be noted that this metaphor represented the access rights of only one subject at any one time. To view a different subjects access rights and current accesses, a separate method was to be provided to switch between users.

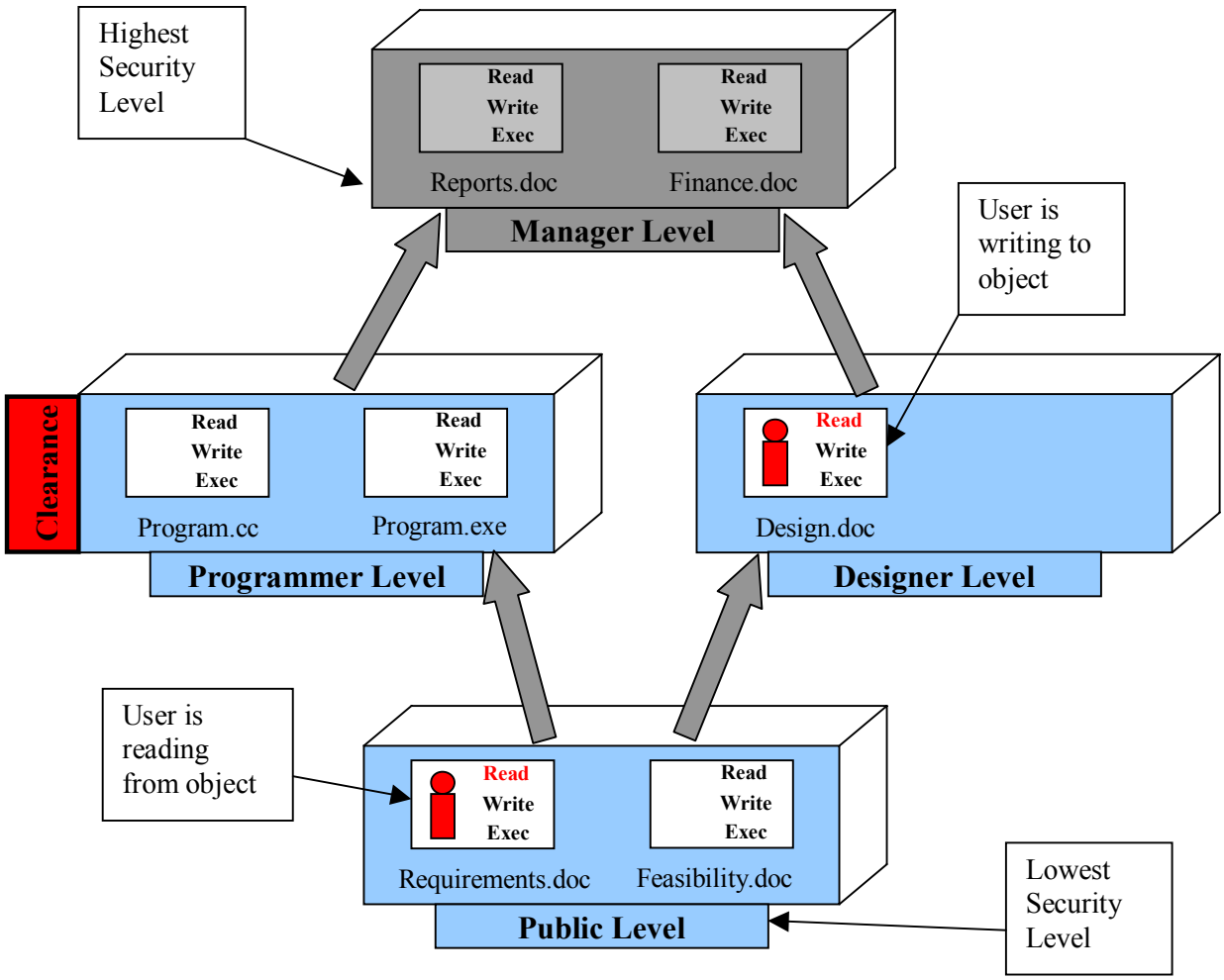


Figure 4.2: Layers metaphor

The BLP Model has 2 main properties. These are the simple security (ss) property and the star (*) property (see section 3.2.1). The metaphor was designed so that it would be easy for the user to see when these properties are violated. The ss-property is a "no read-up" policy as mentioned previously. This states the subject cannot read any objects above the subject's security clearance. A violation of this property should be easy to spot, as the metaphor greys out any layers above the subject's clearance level. Therefore if an operation were performed on an object in a greyed out layer, which was above the subject's clearance, then it would be easy to see that the ss-property has been violated.

The *-property is a "no write-down" policy as mentioned in section 3.2.1. This states the subject cannot **read from** any objects, whose security level is **above** any of the objects that the subject is currently **writing to**. This was more difficult to see through the metaphor. Therefore some explicit feedback was provided for violations of the *-property. Firstly, if a no-write-down occurred, then the write access that had caused the no-write-down, flashed to indicate violation of the *-property. Furthermore, a user could also want to see which accesses may potentially cause a violation of the *-property to occur. Therefore, any access which would cause a violation of the *-property was marked with a red cross (X) drawn over it. This showed the user at any time which accesses would cause a no-write-down. For example, if a user is writing to an object at one level then all the objects which cannot be read as it would be a violation of the *-property would have their read access marked with a cross. Similarly, if a read access was made to an object, then any write access which would violate the *-property would be marked with a cross.

4.4.2 Role Based Access Control (RBAC) Model

4.4.2.1 Main Concepts Revisited

The RBAC model is centred on the notion of roles. Users are made members of roles. Roles have operations assigned that can be carried out by a member of that role. In this model operations are assigned to roles per object rather than per security level as in the last model. Therefore, there is no notion of levels in this model. Therefore, we can see that the RBAC model defines access on an individual object-by-object basis. Similarly, operations are also defined per object. So the main idea is that there is a set of objects, which are accessed through authorised operations via authorised roles.

4.4.2.2 Designing The Graphical Metaphor

The previous metaphor of layers did not apply to this model, as there was not a distinct notion of layers. Using a similar approach as before, we could think of the RBAC model as a grid with each cell representing a table (i.e. database approach). As before we could indicate an access through icons inside the object representing the cell for the object. While this was a valid graphical metaphor, it

would not have been exploiting the users mental model, as the grid metaphor didn't intuitively represent objects and accesses. This is shown in figure 4.3 below.

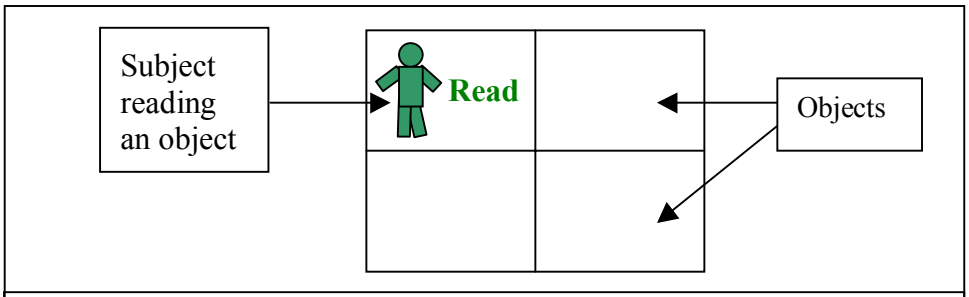


Figure 4.3: Grid metaphor

Extending on the grid approach we could have thought of each object as a box, with users moving in and out of boxes to access tables. This was a slightly better representation but it still didn't feel very intuitive. An extension to this was to use a concept that was more familiar to users. Therefore the boxes were thought of as rooms in a virtual building. These rooms each represented a table in the database. Furthermore, the metaphor was made more comprehensive by differentiating between read and write operations. By using doors subjects were allowed to enter the room when they wanted to write to an object. Then to read from an object the concept of windows into the rooms was used. As subjects activated roles, the appearance of the rooms would change, and subjects would then move around the environment as they accessed objects.

This metaphor may have been a more natural fit with the users mental model. Although most users won't have used a metaphor like this before, it was assumed that they would appreciate the difference between looking through a window and entering a room and how this related to access operations. This metaphor was the better option as it covered all the concepts and was assumed to be a better fit with the users mental model (although this can't be proved until the evaluation stage).

As was mentioned before, this model was thought of in a database environment, therefore there was no concept of executing an object. However, the metaphor could be extended to cover this. For example, a slot in the door to a room could be used, where a user could post something through and then have something returned. This could be a valid concept to represent execution of an object.

Figures 4.4 and 4.5 below show how the metaphor might be visualised. Figure 4.4 below shows a plan view. Each cell represents a room i.e. a table. The object inside the room represents the data. The doors and windows can also be seen for each room. Figure 4.5 below shows how this might look in 3-dimensional form.

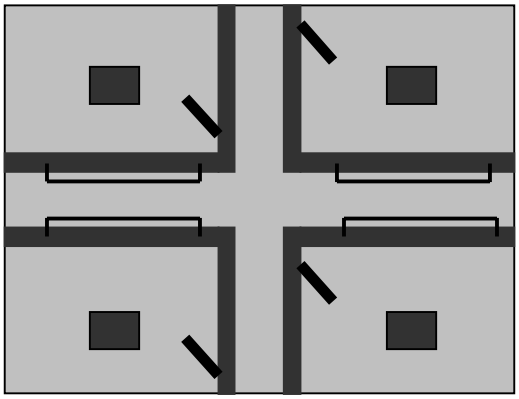


Figure 4.4: Plan view of environment

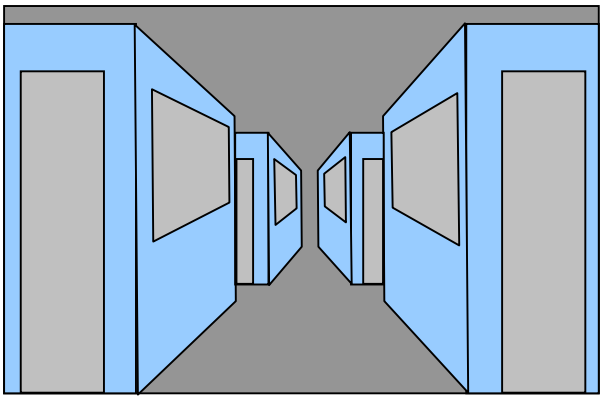


Figure 4.5: 3D view inside the environment

While this metaphor seemed to cover all the concepts, it would have had difficulty representing certain operations in a database environment. For example, if a user issued a SELECT command to read from multiple tables, then this metaphor would have had trouble displaying this. This is because it would have had to show the user going from reading (looking through a window) one room to reading another room. This implied multiple accesses when there would only have been one physical access. This would have been a serious flaw, as it would misrepresent the nature of the access.

To try and rectify this, the previous metaphor was modified into one that would allow multiple reads to be represented as one operation. Allowing the subject to see through more than one window at once achieved this. To do this the environment was rearranged so that all the rooms surrounded the user. This arrangement allowed the user to stay stationary whilst reading objects. Read operations were represented by a line from the user going through the window to the object (line of sight). This is shown in figure 4.6 below. Write operations were represented by moving the subject inside the room. This is shown in figure 4.7 below.

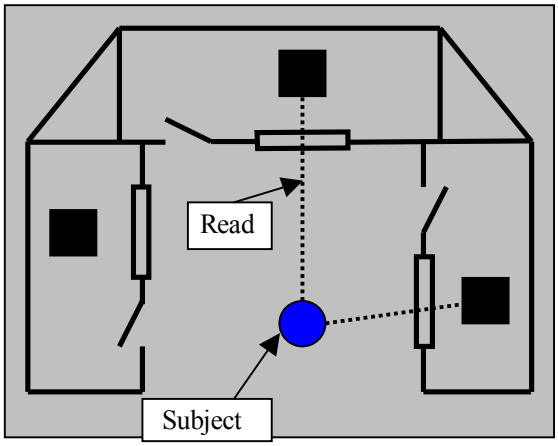


Figure 4.6: Subject reading from 2 objects

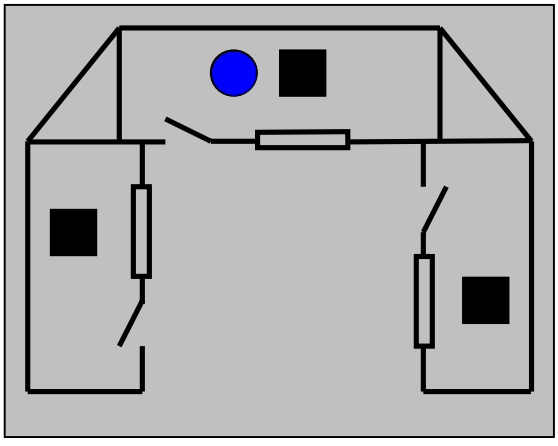


Figure 4.7: Subject writing to an object

Also, this metaphor represented rooms differently depending on the access a subject was allowed through their active role. So if a read were authorised for the active role on an object then a window would be drawn and if a write were authorised then a door would be displayed. However if either of these access rights were not authorised, then the corresponding window or door would not be displayed to represent the fact that a subject was not allowed to look into (read), or enter (write) that room (object).

This metaphor could also be applied to larger systems by making the environment larger and having more rooms in the environment. However, this metaphor may not scale very well to very large databases.

This metaphor fixed the problem of the previous metaphor. However, it could be argued that, a write operation could have a read operation included in it, which would not be represented as one operation by this metaphor. While this metaphor would not have been able to represent such an operation, it is important to note that such a write operation is conceptually two operations, first a read, then a write. Another problem with this metaphor was that, it was assumed that the user could actually see into multiple rooms, which were in front, and to either side. In a real situation a human may have to move their head to see through these multiple windows. This could be a weakness in the metaphor, as it does not actually reflect reality.

Whilst this metaphor sufficed for a database environment, it would not be suitable for an environment with shared files, such as the one used for the BLP model previously. This is because this metaphor cannot represent writing to two files at once or reading from one file whilst writing to another etc.

The RBAC model being studied for this project [16], introduces many extra constraints/rules to the simple notion of users, roles and operations to provide a much more comprehensive security model. Each of the rules discussed in [16] are now analysed further to establish whether or not they could be integrated into the graphical metaphor described above. If they could be integrated then an attempt was made to design graphical displays to represent these rules.

Rule 1 – Role Hierarchy

This rule states, "if a subject is allowed to access a role and that role contains another role, then the subject is also allowed to access the contained role" [16]. This rule creates a hierarchy of roles, where some roles contain other roles to form a tree-like structure. This tree like structure is a convenient way of representing these roles. The tree structure is used quite often in computer applications and a well-known example is the Microsoft Windows Explorer Tree. A tree to represent role hierarchies would be similar to the Windows Explore tree and would be represented as in figures 4.8 and 4.9 below.

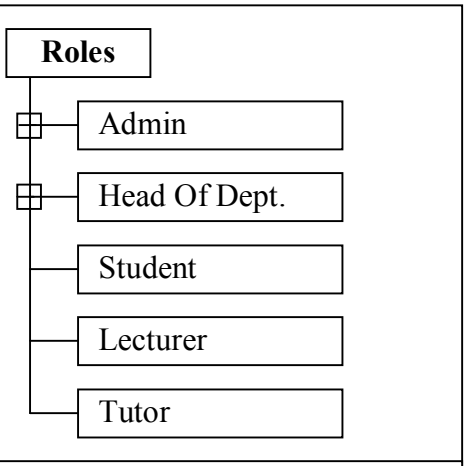


Figure 4.8: Collapsed tree for role hierarchy

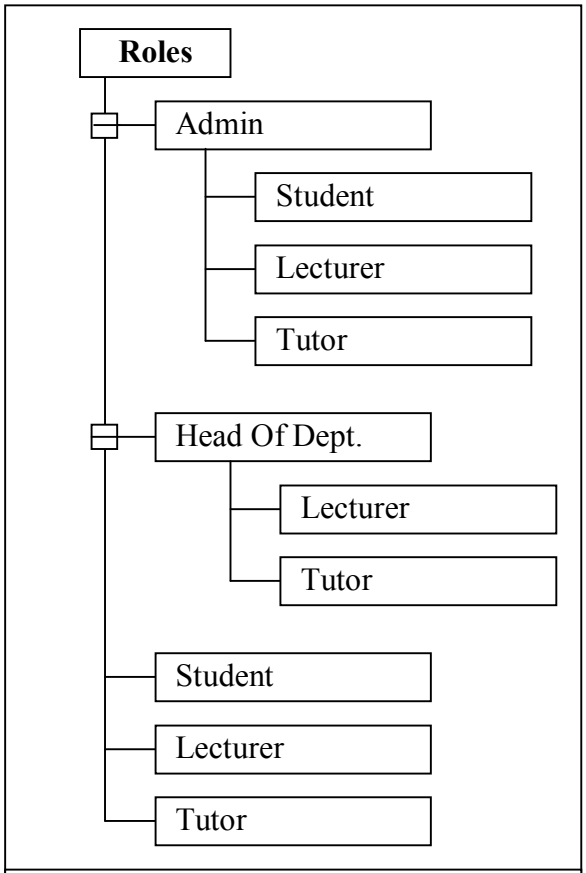


Figure 4.9: Expanded tree for role hierarchy

Figure 4.9 indicates that the Head of Dept. role contains the lecturer and tutor roles. As a starting point this tree structure could simply provide a visual representation of the role hierarchy. However, this concept could also be extended to allow the roles to be displayed in different colours depending on their status i.e. whether they were active, mutually exclusive, unauthorised, over the limit etc.

Rule 2 – Static Separation of Duty

This rule relates to the stage when subjects are assigned to be members of roles i.e. when the administrator assigns the role to the user. This deals with conflicts of interest at the time of granting a user membership of a role and says that a user cannot be made a member of a role if this role is mutually exclusive with another role, which the user is a member of. This could be represented by having a display to show the roles that have been authorised for a user and the roles, which could still be authorised, with mutually exclusive roles greyed out. The actual roles could be represented by graphical job tags, as these are used in the real world to indicate a person's job/role in a business. This is shown in figure 4.10 below, where the user is authorized for the Head of Dept and Lecturer roles and these roles are mutually exclusive with the Tutor and Postgraduate roles (greyed out).

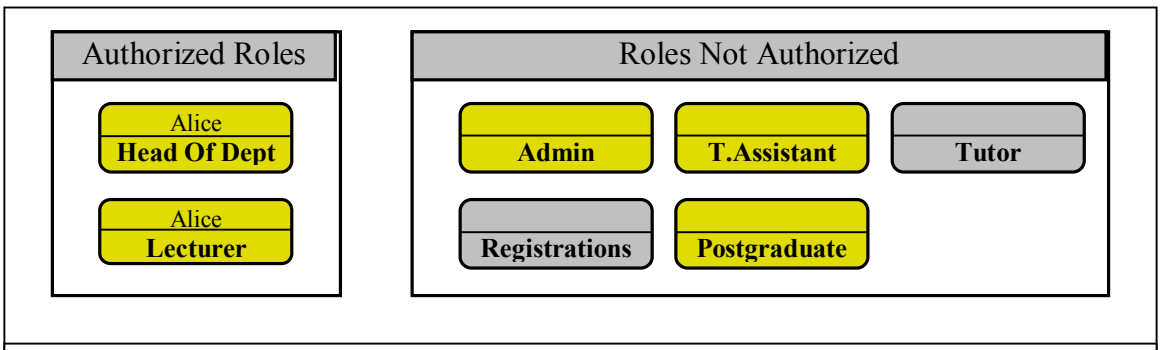


Figure 4.10: Static Separation of Duty metaphor

Rule 3 – Cardinality

This rule states that there may be some upper limit for a role. This upper limit could apply at the stage when roles are assigned to subjects or at the stage when a subject activates a role. For this project the latter is assumed. Therefore there may be a limit to the number of users that may be active in a role at any one time. This is a two-part rule, which deals with the limit a given role may have and the number of users that are currently active in that role. This could be represented by a visual icon which showed how many users were active in a certain role as well as how much the limit is for the role. This visual icon could be represented as a type of progress bar as in figure 4.11 below.

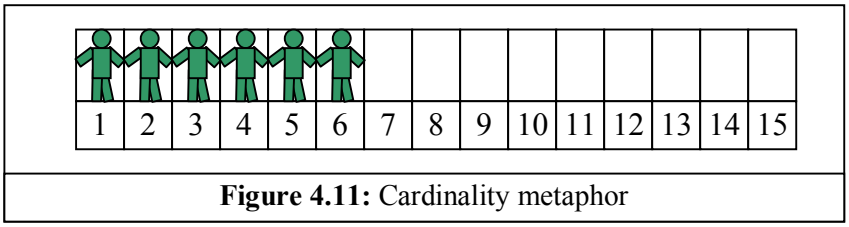


Figure 4.11: Cardinality metaphor

The example in figure 4.11 shows how the cardinality metaphor would look for a role with a cardinality of 15 in which 6 users were active. This metaphor would only need to be displayed if a role actually had an upper limit specified. If it didn't have a limit specified then there is no need for the metaphor as the purpose is to represent the cardinality of a role. This metaphor could extend both horizontally and vertically (downwards), to represent a larger upper limit if necessary.

Rule 4 – Role Authorization & Rule 5 – Role Execution

These 2 rules are similar in that they both relate to role activation. Rule 4 says "a subject can never have an active in a role that is not authorised for that subject" [16]. This means that a subject can only activate authorised roles. Rule 5 says "a subject can execute an operation only if the subject is acting within an active role" [16]. This refers more to carrying out operations but it means that the subject must activate a role before they can execute the operations allowed for this role. Note that the notion of activating a role is being treated as something the subject will explicitly carry out i.e. a login for each role, as opposed to something that is implicit. To represent the idea of activating roles, the previous metaphor of job tags could be used to show which roles are active. The display could show all the roles that a subject is authorised for and could highlight the active roles with a green background. The non-active roles would have a yellow background (the red background is explained in the next rule). This is shown in figure 4.12.

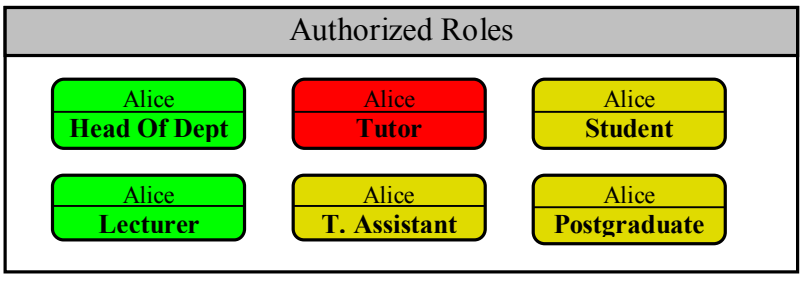


Figure 4.12: Role Activation Metaphor

Rule 6 – Dynamic Separation of Duty

This rule says a subject cannot activate a role if this role is mutually exclusive with another role, which is already activated. Again, like rules 4 and 5, this rule relates to activating a role. This is because this rule will come into effect when a role is activated. Therefore, the previous metaphor for role activation could be extended further to display the change when a role goes from being available for activation, to being mutually exclusive and not activatable. Therefore, if an active role was mutually exclusive with another role that the user was authorised for, then the mutually exclusive role could be highlighted red. This could be displayed as shown in figure 4.12 above for the tutor role, which may be mutually exclusive with the lecturer role, and is therefore highlighted red.

Rule 7 – Operation Authorization

This rule says "a subject can execute an operation if it is authorised for the role in which the subject is currently active" [16]. This rule would be represented by the metaphor of rooms, doors and windows described previously (figures 4.6 & 4.7). The rooms represent the objects, and the windows & doors (or lack of them) would represent the operations available to the subject. So if a subject was looking into a room without a window or was inside a room without a door then the subject was obviously breaking the rule. Furthermore, the rule mentions the currently active role, so the metaphor would change every time the user requested to see a different active role. This is discussed in chapter 5.

Rule 8 – Operational Separation of Duty

This rule relates to business functions rather than simple modes of access. This rule is provided to ensure that certain business functions cannot be carried out by a single role. This is done by ensuring that the sub functions of this overall function cannot all be carried out by one role. This rule relates to specific organisational policies and will vary widely between organisations. This could be represented by a task list for each overall function. The task list could be represented in the form of a clipboard. On the clipboard could be the sub functions of the overall function. This task list will have a tick box next to each sub function. If a role is authorised to carry out the sub function, then there will be a tick next to the sub function, otherwise the box will be empty. Obviously the ticks will change depending on which role the user of the metaphor is viewing. For this rule to be satisfied, there should never be a clipboard where all the boxes are ticked. An example is shown in figure 4.13.

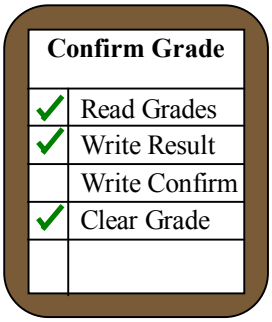


Figure 4.13: Operational Separation of Duty Metaphor

Rule 9 – Object Access Authorization

This rule is a combination of some of the other rules specified previously. This says that a subject can carry out an operation on an object only if the subject is active in a role (Rule 5), the operation is authorised for this role (Rule 7), and the object is accessible through this operation (it may not be possible to perform a certain operation on an object i.e. executing a text file). This is represented through the combination of the metaphors described previously.

4.4.3 Applicability of the Metaphors to Other Security Models

The metaphors designed previously were tailored towards the features of the BLP and RBAC models. However, it may be possible to alter these metaphors to allow them to represent the other security models studied during this project i.e. the Harrison-Ruzzo-Ullman (HRU) and Chinese Wall Models.

The HRU model is based around access rights that are constantly changing and subjects and objects that are added and removed. The HRU model is similar to the BLP and RBAC models in that the subjects, objects and operations are similar in the models. However the HRU model does not use security levels or roles, therefore the previous metaphors are not easily adapted to the HRU model.

If the concept of levels is ignored from the BLP model metaphor and the objects are represented in the form of a grid then it may be possible to use a similar metaphor for the HRU model. Obviously, the metaphor would need to be adapted to allow access rights to be granted/denied and the addition/removal of subjects and objects must be facilitated. However, while this metaphor may work, it is not very graphically informative, in terms of the metaphor actually representing the model. Similarly the RBAC model could be adapted so that instead of having static rooms, the user of the software could grant and deny access, thereby changing the appearance of the rooms i.e. adding/removing windows and doors. Also adding and removing objects could change the number of rooms. This may be a better metaphor, as it would allow the user a richer graphical interface, which would be more descriptive and may be easier to understand.

The Chinese Wall (CW) model is different to the BLP and RBAC models. This model deals with the concept of conflicts of interest between “companies”. The basic concept of the Chinese Wall model is that all objects that concern the same company are grouped into datasets. These datasets are then grouped into conflict of interest classes, such that each class contains datasets of companies that are in competition with each other. The rule, which then governs access to the classes, says that subjects can only access at most one dataset from each conflict of interest class.

It is not obvious how the previous metaphors could be adapted to represent the Chinese Wall (CW) model. Referring back to the comparison made in section 3.2.5, the main difference between the BLP model and the CW model is that the BLP model uses layer whereas the CW model uses conflict of interest classes. Therefore, instead of using the layers in the BLP metaphor as security levels, they could be used as conflict of interest classes. Within each layer representing a conflict of interest class, all the company datasets belonging to that class could be displayed in the same way that an object was displayed in the BLP metaphor. Then when a subject accesses a company dataset, all the other company datasets in that conflict of interest class could disappear and the one that was accessed could be displayed by itself. This is one possibility, but obviously it would have to be ensured that the layers do not imply that the conflict of interest classes are somehow ordered.

The RBAC metaphor could also be modified to represent the CW model. As mentioned in section 3.2.5, the RBAC model uses the concept of roles whereas the CW model uses conflict of interest classes. This doesn't help much as it doesn't imply an obvious way of altering the metaphor. However, if the objects in the RBAC metaphor, i.e. the rooms, were considered as conflict of interest classes then it may be possible to modify the metaphor. The rooms could be empty to start with, and then when a subject actually accesses a company dataset from a conflict of interest class, the room representing that conflict of interest class could have the company dataset placed inside it. To accompany this metaphor, another metaphor would probably be needed to allow the user to see which company datasets are in which conflict of interest class. The ideas presented in this section could work if they were implemented. However, with all of the ideas presented in this section, further design work would need to be done to investigate whether these metaphors are actually feasible and to determine the specifics of how they should be designed.

4.5 Summary

In this chapter, a methodology was chosen to design graphical metaphors and this methodology was used to design various metaphors for the BLP and RBAC security models. Furthermore, some analysis was also done to determine whether the metaphors designed in this chapter could be extended to support the other security models discussed in chapter 3. In the next chapter, the interface design for the software will be presented. This interface will integrate the metaphors designed in this chapter with several user controls and other aspects, which will be needed to allow the metaphor to be manipulated and for a full application to be developed.

Chapter 5 – Interface Design

5.1 Introduction

In this chapter, the designs of the metaphors from the previous chapter are combined together with a user interface to form a full software application. To do this, a step-by-step approach was used to, firstly, design the user interface. Then, secondly, to integrate this interface with the metaphors described previously to form a single application. In this chapter the process of designing the interface for this application is documented. Firstly, a description is provided of how the software works. After this, the user interface controls required to achieve the functionality of the software are described. Then finally, a design is presented that combines the user controls described in this chapter with the metaphors described in chapter 4. It should be noted that while the final design is presented here, the design was produced over a period of time, after several iterations of the design process.

5.2 The Bell-LaPadula (BLP) Model

5.2.1 Functionality of the Software

The software for the BLP model aimed to simulate subjects making accesses and applied the properties/rules of the BLP model to these accesses. The software reads in some details, from files set up by the user, to set up the model. These files detail the subjects, objects and the access rights of the subjects on the objects. It also reads in an access log set up by the user before starting the software.

The software then displays the objects it has read in, through the metaphor of "layers" as described in section 4.4.1. The display of this metaphor will depend on which subject the user has chosen to view. Then the software reads in accesses from the access log, one access at a time and displays these on the metaphor. The user of the software will only notice an access has been made if they are viewing the subject who has actually made the access. If an access violates the *-property then this is explicitly highlighted in the metaphor.

In addition to the access log, it was decided that extra functionality would be provided to allow users to define new accesses whilst the software is running. This was introduced, as it seemed unreasonable to have to define all the accesses you wanted to make in advance of starting the software. Furthermore, if the user wanted to try something they hadn't thought of before, then they would have to quit the program, rewrite the access log and start the program again.

Another extra that was provided was a display that showed what had been read in so far from the access log and was updated every time a new access was read in from the access log. This was provided, as it was difficult for users to see what had been read in from the access log, especially if it was for a subject other than the subject that the user was viewing at the time.

5.2.2 User Controls Required

From the description in section 5.2.1 it could be seen that several user controls would need to be provided to allow users to use the software. The user controls that were provided were: -

- A way to change the subject the user is currently viewing. This allows the user of the software to switch between the subjects and see what different subjects are doing. This was implemented with a drop down list.
- A way of adding new accesses to the metaphor. This control allows users to define new accesses while the software is running. For this the user will need to specify the subject making the access and the object being accessed. These were both implemented using drop down lists. Furthermore, the type of access being made also needs to be specified i.e. read, write, execute or release. This was implemented using a checkbox for each type of access as more than one may need to be selected. Also a button was implemented to confirm the access.
- A way to allow the user to specify whether or not to allow downgrading of subjects to write to objects below their clearance level. This was provided as it is part of the *-property of the model and is ignored if the subject is trusted. A checkbox was used to implement this feature.
- A way to specify blind writes. This allows users to define whether a write access is to be regarded as a blind write or whether it also allows the subject to read from the object. This was implemented as a checkbox. (Note: activating this affects all current write accesses).
- Basic functions to allow the user to pause the software (so that accesses aren't read in form the log) and to step the reading of accesses from the log were also implemented as buttons.

5.2.3 Integrating the User Controls with the Metaphor

From the analysis done in sections 5.2.1 and 5.2.2 and the metaphor designed in section 4.4.1, it can be seen that all the elements required for the software have now been discussed. In this section these elements are joined together and presented as one overall interface. From the work done so far, three main elements of the software can be identified. Firstly, the metaphor of layers to display the objects, accesses, rights etc. Secondly, the user controls to manipulate the metaphor and finally the display representing the access log. The interface design chosen for this software is shown in figure 5.1 below. This interface was designed by following the interface design guidelines presented in [22]. These guidelines emphasise the point that the interface should not be too complex, should behave as expected and should have a consistent look and feel.

This design shows that the metaphor takes up the majority of the display. The user controls have been docked to the right side of the window. The user controls were joined to the window rather than left floating, as it seemed neater and more compact to do it this way. The access log display is docked at the bottom of the screen and this was done, as it is usual to see status bars, progress bars and other similar displays at the bottom of the window.

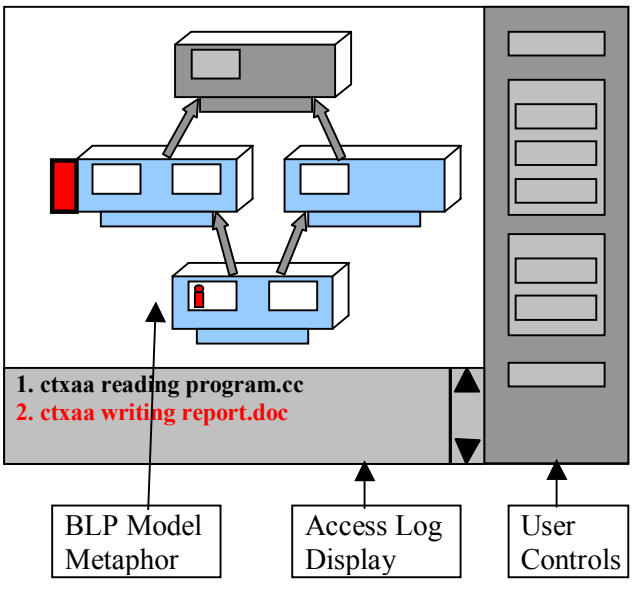


Figure 5.1: BLP Model Software – Interface Design

5.3 The Role Based Access Control (RBAC) Model

Before discussing the design of the interface for the RBAC software it is important to discuss some of the design decisions that have been made. In section 4.4.2, several metaphors were designed to represent the different rules of the RBAC security model. It did not seem sensible to aim to implement all of these metaphors as there wasn't enough time during the project and also because not all the metaphors were needed to demonstrate the effect of using a metaphor to demonstrate the RBAC model. Therefore, a decision was made to implement only the metaphor of rooms with windows/doors to represent the security model. This meant that the metaphors discussed to represent the rules of the model i.e. cardinality, separation of duty, role activation, role hierarchy etc. were not implemented. Therefore the design in the next few sections discusses the software that was actually implemented i.e. not including the metaphors for rules. However a design has been proposed that would include all the metaphors described in section 4.4.2. This design is presented in Appendix C.

5.3.1 Functionality of the Software

The software for the RBAC model aimed to simulate subjects making accesses and applied the properties/rules of the RBAC model to these accesses. However, unlike the software for the BLP model, the RBAC software also allows users to change a subject's authorised roles and a role's authorised rights. The software reads in some details, from files set up by the user, to set up the model. These files detail the subjects, roles, the role hierarchy, the access rights of the roles on objects and authorised roles for each subject. It also reads in an access log set up by the user.

The software displays the details it has read in, through the metaphor of "rooms" as described in chapter 4, section 4.4.2. The user of the software must make a choice as to which user/role

combination they would like to view. Then the metaphor displays the rooms depending on the access rights of the chosen role and whether or not the chosen subject is authorised and has activated this role. After this the software reads in accesses from the access log, one access at a time and displays these on the metaphor. The user of the software will only notice an access has been made if they are viewing the subject who has actually made the access.

As for the BLP model, it was decided that extra functionality would be provided to allow users to define new accesses whilst the software is running. Also a display showing what had been read in so far from the access log was also provided. These were introduced for much the same reasons as explained above for the BLP model (see section 5.2.1). As mentioned before, the software also allows a user to edit the role hierarchy, the rights of a role and also the roles a user is authorised for. These facilities allow users to modify the state of the system whilst the software is running. These facilities were provided as they allow users to interactively change the rights a subject may be allowed and allow users to see the effects of these changes represented on screen through the metaphor of rooms.

5.3.2 User Controls Required

From the description in section 5.3.1 it could be seen that several user controls would need to be provided to allow users to use the software. The user controls that were been provided were: -

- A way to change the subject and role the user is currently viewing. This allows the user of the software to switch between subjects and view the rights/accesses of subjects through various roles. This was implemented with a drop down list for each of the subject and role options.
- An interactive method of activating and deactivating roles. Whilst having an entry in the access log to activate/deactivate a role facilitated this, it was also a nice feature to allow this interactively. This control allows the user to activate/deactivate the currently viewed role for the currently viewed subject. This was implemented through simple push buttons.
- A way of adding new accesses to the metaphor. This control allows users to define new accesses while the software is running. For this the user will need to specify the subject making the access and the role the access is made through. These will both be taken as being the currently viewed subject and role. The user will also need to specify the object being accessed and the type of access being made i.e. read/write. These were both implemented using drop down lists. Also a button was implemented to confirm/submit the access.
- A way of changing the role hierarchy. This control allows users to define the roles contained within each role. For this the user will need to specify the role for which the “contains” relationship is being edited. This was implemented as a drop down list. The user will also need to specify which roles the selected role contains. This was implemented as a set of

checkboxes, one for each role, as more than one role may need to be checked. It was decided to implement the overall control of altering the role hierarchy as a floating sub window, which can be opened by clicking on a button. This was done in this way as this control took up a lot of space and would have made the main interface look cramped.

- A way of changing the rights allowed for a role on each object. This control allows users to dynamically modify what a role is allowed to access. For this the user will need to specify the role for which the rights are being edited. This was implemented as a drop down list. The user will also need to specify the rights of this role on each object. This was implemented as a pair of checkboxes for each object, one for read and one for write. Again this was implemented as a floating sub window.
- A way of changing the roles each subject is authorised for. This control allows users to dynamically modify which roles a subject is allowed to activate. For this the user will need to specify the subject and role combination, which is being authorised. This was implemented as a pair of drop down lists. The user will also need to specify whether this role is authorised for this subject. This was implemented as a single checkbox. Again this was implemented as a floating sub window.
- Basic functions to allow the user to pause the software (so that accesses aren't read in form the log) and to step the reading of accesses from the log were also implemented as buttons.

5.3.3 Integrating the User Controls with the Metaphor

From the analysis done in sections 5.3.1 and 5.3.2 and the metaphors designed in section 4.4.2, it can be seen that all the elements required for the software have now been discussed. In this section these elements are joined together and presented as one overall interface. As mentioned before, not all the metaphors from section 4.4.2 were implemented and thus they are not included in this design. Again this interface was designed by following the interface design guidelines presented in [22]. The interface design chosen for this software is shown in figure 5.2. This interface includes the metaphor of rooms, a display showing the access log and also the majority of the user controls. This interface was designed to look similar to the BLP model software shown in section 5.2.3.

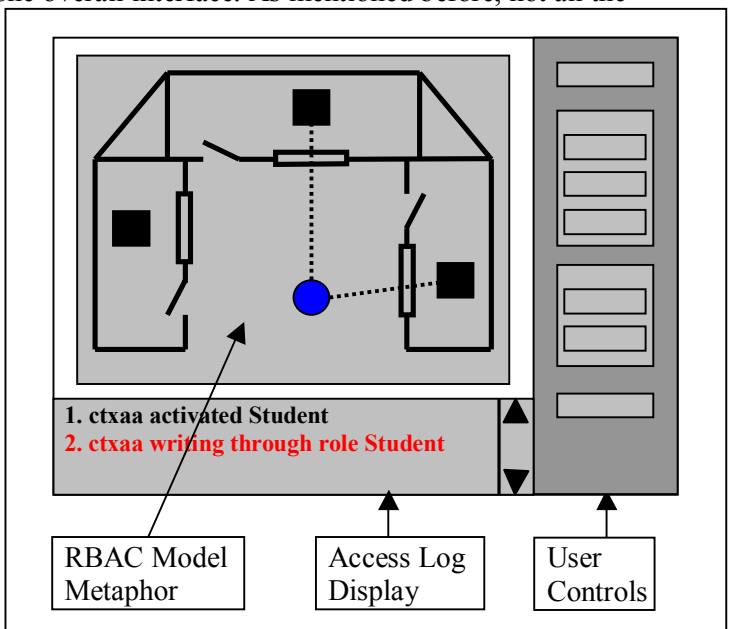


Figure 5.2: RBAC Model Software – Interface Design

The user controls were docked to the side of the window. However these user controls did not include all the user controls described in section 5.3.2. The functionality to change the role hierarchy, change the rights for each role and to change the authorised roles for a subject were implemented using floating windows. These functions require a lot of controls, which would have cramped up the main interface. These windows are described in the next paragraph.

Figure 5.3 shows the design for the window that allows the user to alter the role hierarchy. This window allows the user to select a role. Once a role is chosen, the user then ticks the roles, which are contained within the selected role. Figure 5.4 shows the design for the window that allows the user to alter the rights for each role. Again the user selects a role using the drop down box, and then the user could change the read and write access rights on each object for this role through the checkboxes provided. Figure 5.5 shows the design for the window that allows the user to alter the authorised roles for each subject. The user selects a subject/role combination through the drop down boxes provided. Then the user could tick the "Authorised?" checkbox if the role were authorised. The "Contained?" checkbox shows whether or not the role is authorised for this subject by being contained in another role. The "Contains?" checkbox was implemented so that it was not modifiable (only viewable).

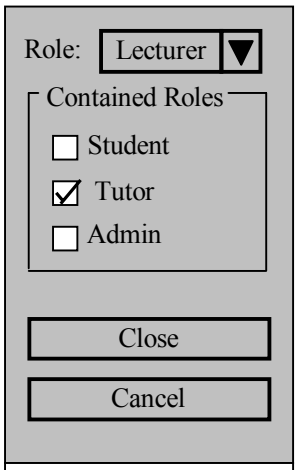


Figure 5.3: Pop up window to change contained roles

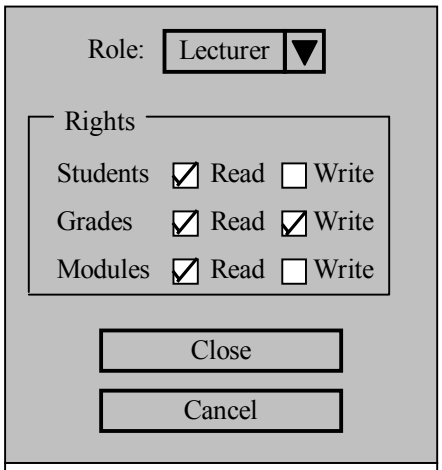


Figure 5.4: Pop up window to change access rights for each role on each object

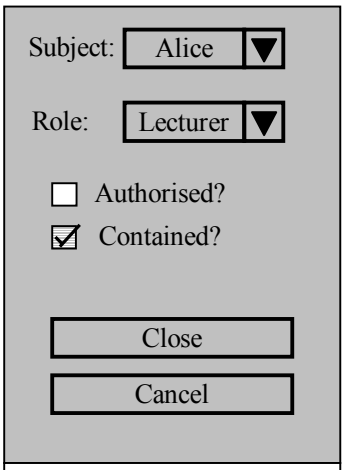


Figure 5.5: Pop up window to change authorised roles

5.4 Summary

In this chapter the metaphors designed in chapter 4 have been integrated into a user interface to form a design for a complete application. In the next chapter the structure of the program will be discussed and the design for some of the major functions of the software will be presented.

Chapter 6 – Code Design

6.1 Introduction

In this chapter, an overview is given of the main structure of the code used for the Bell-LaPadula (BLP) and Role Based Access Control (RBAC) programs. For each program, the major classes used are described. Following this the functions for reading from files are described. Then finally, the main drawing functions are discussed.

6.2 The Bell-LaPadula (BLP) Model

6.2.1 Class Design

For the BLP model five classes were used. It is important to note that the classes used in this program merely contained some public variables and did not contain any class operations. Therefore C-style structs were used rather than C++ classes (but they will still be referred to as classes).

These classes were derived from the main concepts used in the model. Firstly a "Subjects" class was used. Objects of this class contained details of the users in the system and held the subject ID, subject name and the clearance of the subject. Similarly an "Objects" class was also used. This contained details of the resources in the system and held the object ID, object name and the classification of the object. An "AccessMatrix" class was also used. This linked together the subjects and the objects and defined the exact access rights that were authorised for a subject on an object. This was a matrix that contained 3 variables representing the read, write and execute rights. To reference an element in this matrix a subject ID and an object ID had to be provided.

Two further classes were also used. These classes were related to the accesses made during the program run. One of these classes was the "AccessLog" class. This class stored the access log and defined all the accesses that had to be made during the simulation. Objects of this class held a subject ID, an object ID and the access made. The other class used was the "CurrentAccesses" class. This class stored all the accesses currently being made in the system.

6.2.2 Reading From Files

For the BLP program there were many details that were read in from files. For all the files that were read, the same code was used. This code is presented in figure 6.1. For the BLP program the Subjects, Objects, Access Rights (Access Matrix) and the Access Log were all read in from files. The details extracted from each line would be different for each of these files.

1. *Open the file*
2. *Read in a single line from the file*
3. *Extract the relevant details from this line*
4. *While first item extracted != -1*
    5. *Read in another line from the file*
    6. *Extract relevant details from this line*
7. *Loop*
8. *Close the file*

Figure 6.1: File reading function

6.2.3 Drawing Functions

In this section, some of the main drawing functions used in the BLP program are discussed and some pseudocode is presented to demonstrate the implementation of these functions.

DrawLayerBox Function - This function was used to draw the container, which represents the concept of a layer and which holds the objects that are present in the layer.

If the currently viewed subjects clearance allows access to this layer
*Draw the layer container in **blue (accessible)** and write the layer name in **black***

Else if the subject is not allowed access
*Draw the layer container in **grey (not accessible)** and write the layer name in **grey***

DrawObject Function - This function was used to draw an object into a specified layer. This function checks whether the object is accessible to the currently viewed subject and draws the object accordingly. This function also checks to see if the current subject is accessing the object. If the object is being accessed, this function changes the appearance of the object to show this access.

If the currently viewed subject is allowed access the layer in which the object is being drawn
*Set a Boolean variable **accessible** to **TRUE***
If there is an object below the object being drawn, that is being written to by the viewed subject
*Set a Boolean variable **writeBelow** to **TRUE***
If there is an object above the object being drawn, that is being read from by the viewed subject
*Set a Boolean variable **readAbove** to **TRUE***
If the object being drawn is being written to by the viewed subject and this write violates the No-write down property
*Set a Boolean variable **noWriteDown** to **TRUE***

*If **accessible** is true draw the object **green** else if **accessible** is false draw the object **grey***

If subject is allowed read access according to access control matrix
*draw read label **white***
*Else draw read label **grey***

If subject is allowed write access according to access control matrix draw
*write label **white***
*Else draw write label **grey***

If subject is allowed execute access according to access control matrix
*draw execute label **white***
*Else draw execute label **grey***

*If **writeBelow** is true draw a cross on the read label*
*If **readAbove** is true draw a cross on the read label*
*If **noWriteDown** is true make the write text flash*

If the viewed subject is accessing the object being drawn
*Draw a subject icon on the object and highlight relevant access text **red***

DrawLayer Function - This function was used to draw the entire layer i.e. the layer box and also all the objects within the layer. The objects to be drawn within the layer were passed into this function.

Translate to the correct position and call the DrawLayerBox function

Create a new viewport (drawing area) within the layer box that has just been drawn

Do an initial translation within this viewport using a variable called ***scroll***
// this allows objects drawn, to be moved, by changing **scroll** to make it look like they're scrolling

For all the objects to be drawn in this layer
    Call the DrawObject function to draw the object within this viewport
    Then translate further across the viewport to draw the next object in this layer

DrawHistoryOfAccesses Function - This function was used to draw the section of the window that displays all the accesses that have been read in from the log so far.

Draw the background display and some scroll buttons for the access log display box

Do an initial translation within this viewport using a variable called ***scrollLog***
// this allows the display to be moved by changing **scrollLog** to make it look like they're scrolling

For all the accesses that have been read in from the access log so far
    Draw onto the screen the details of the access made
    Then translate further across to draw next access record

If all the accesses from the access log have been read in
    Draw simulation finished into the access log display box

Display Function - This function was used to draw the entire metaphor i.e. all the layers and objects, the display to show the access log etc.

Translate to the correct position and draw a red box to show the subjects clearance level

For each of the 4 layers call the DrawLayer function passing the correct objects

Draw the arrows between the layers

Create a new viewport (drawing area) at the bottom of the window

Call the DrawHistoryOfAccesses function to draw the access log display within the viewport

6.3 The Role Based Access Control (RBAC) Model

6.3.1 Class Design

For the RBAC model four classes were used. Again it should be noted that they only contained some public variables and not any operations and so C-style structs were used. These classes were derived from the main concepts used in the model. Firstly a "Subjects" class was used. Objects of this class contained details of the users in the system and held the subject ID, subject name, the roles the subject is authorised for and the roles the subject is actually active in. A "Roles" class was also used. This

contained details of the roles in the system and held the role ID, role name, the rights of the role and the other roles the role contains (role hierarchy).

Similar to the BLP software, two further classes were also used. These classes were related to the accesses made during the program run. These classes were the "AccessLog" class and the "CurrentAccesses" class. These classes performed the same functions as described for the BLP software in section 6.2.1.

6.3.2 Reading From Files

For the RBAC program the same file reading code was used as described in section 6.2.2. For the RBAC program the Subjects, the Roles, the roles authorised for each subject, the Access Rights for each role, the Role Hierarchy and the Access Log were all read in from files.

6.3.3 Drawing Functions

In this section, some of the main drawing functions used in the RBAC program are discussed and some pseudocode is presented to demonstrate the implementation of these functions.

DrawRoom Function – This function was used to draw a single room given a name and an ID. The function checks the accesses that are allowed for the currently selected subject, role combination and draws the room accordingly.

```
Check the access allowed for the currently selected subject/role combination

If no access is allowed draw a room without a door or a window
If only read access is allowed draw a room with a window but no door
If only write access is allowed draw a room with a door but no window
If both read and write access is allowed draw a room with a window and a door

Draw the name of the room and a rectangle inside the room to represent the resource
```

DrawSubject Function – This function was used to draw an icon to represent the subject given the subjects position. The subject's position relates to the accesses they make and was calculated elsewhere and passed into the function.

```
Translate to subject's correct position and orientation

Draw the icon for the subject at the correct position
```

DrawReadLine Function – This function was used to draw lines from the subject's position to each object the subject is reading (This did nothing if the subject was writing to an object).

```
For each object in the system
        If it is being accessed by the current subject/role combination
                Draw a line from the subject's position to the object (i.e. room window)
```

DrawBoxPanel Function – This performed the same function as the DrawHistoryOfAccesses function described for the BLP model. It used the same logic and code as described in section 6.2.3.

DrawBuilding Function – This function was used to draw the whole collection of rooms and to organise them all around each other.

```
If the viewed role is not activated or not authorised for the currently viewed subject
        Display a message indicating what has occurred

Else if the viewed role is authorised for and activated by the currently viewed subject
        For each object in the system
                Translate and rotate to the correct position for the room
                Draw the room to represent the object by calling the DrawRoom function
```

Display Function – This function was used to put together the DrawBuilding function and the DrawBoxPanel function. It also determines the subject's position in relation to any accesses made.

```
Create a viewport (drawing area) to draw the main metaphor of rooms

Call the DrawBuilding  function to draw all the rooms within this viewport

Check whether the subject is making an access and calculate the position of the subject icon from
the access it is making (if any)

Call the DrawSubject function passing the subjects position

Call the DrawReadLine function to draw the lines to represent any read accesses

Create another viewport (drawing area) at the bottom of the window

Call the DrawBoxPanel function to draw the access log display within this new viewport
```

6.4 Summary

In this chapter an overview has been provided of the structure of the software and the functions used to code the applications. In the next chapter, some of the problems encountered during the implementation process are discussed and a description is given of the issues that arose from the testing carried out on the applications

Chapter 7 – Implementation and Testing

7.1 Introduction

In this chapter, an overview is given of the some of the main issues that were encountered whilst implementing and testing the programs that were developed for the project.

7.2 Implementation Issues

File Reading Functions – Some problems were encountered when writing the file reading functions for the software. It had been planned to use C++ file reading functions within the software. However, due to some difficulties, which seemed to be due to some OpenGL/compiler related problems, these functions did not work as intended. Therefore it was necessary to use C-style file reading functions. Since these functions were familiar, there wasn't too much of a problem.

User Interface Issues – To use the GLUI functions a static library had to be built and included with the source code. Without this static library, the code would take a long time to compile. This static library is fairly large in size but it is necessary to avoid long compilation times.

– Another issue that was discovered with the GLUI interface was that using floating windows could cause problems. This is because after having launched a floating window, if you close it by clicking on the 'X' button at the top right corner of the window, then it not only closes the floating window but also closes all other windows and quits the application. The best course of action here was to provide a custom close button and advise users not to use the 'X' button.

OpenGL Issues – There were some major problems regarding the resizing of windows with OpenGL applications on the School of Computing Windows 2000 machines. It seems there was some sort of problem with OpenGL applications becoming unresponsive if the window became larger than a certain size. After making several enquiries with support staff, there seemed to be no way around this as it was thought to be a conflict between OpenGL and the graphics cards installed on the machines. Therefore, a function was written to ensure the window never became large enough to cause such a problem. While this was unfortunate, it shouldn't cause any real problems as the problem only occurs if the window is made extremely large (more than about 1200x900 pixels).

– The OpenGL coordinate system also caused some unforeseen problems. When trapping mouse clicks in OpenGL applications, the coordinate (0, 0) is located at the top right hand corner of the screen. This is different to conventional coordinate systems where (0, 0) is located at the bottom left corner. To overcome this, the coordinates were altered when mouse clicks were processed.

7.3 Testing

This section describes some of the main issues that arose from the testing process and how these were fixed. Much of the testing done during the project was carried out as the software was being developed. For a full test plan see Appendix D.

Reading From Files – The function failed to recognise whitespace at the end of a file (EOF) and kept waiting to read the next line. Therefore, the file was read until a –1 was reached rather than the EOF.

Scrolling (for BLP objects) – The scroll limit was set wrong. This meant that the scroller did not allow all the objects to be viewed. Therefore the scroll limit was altered.

Drawing Text – Regular GLUT bitmap fonts seem to have problems dealing with changes in window size and their position and orientation went wrong. Therefore GLUT stroke fonts were used which scale with the window and can be manipulated more easily.

User Interface – Using live variables in GLUI didn't seem to work at first but it was because they were not being synchronized in the idle callback. Once this was done they worked properly.

Deactivating Roles – When deactivating roles, the accesses that were still in progress were not lost. So the next time the subject activated that role, the accesses were still there. This was changed.

Editing Authorised Roles – Taking away authorisation for a role which was both authorised explicitly and contained within another role which was authorised for the user, led to the role being denied. This was not the intention of this facility and therefore this was changed so that the role would remain authorised if it were still contained in another authorised role.

7.4 Important Information

Some important information regarding the software that has been implemented is provided in this section. Firstly, for the BLP software, the layers are still hard coded. This means that you can't extend on or introduce new layers. Similarly for the RBAC software, the objects are hard coded. This means that users can only work with these objects. This was the intention from the start of the project as the software was supposed to be experimental software rather than something to be put to proper use. Furthermore there are also some other important limits in the software. In the BLP software there is a limit of a maximum of 100 subjects, 100 objects and a 1000 accesses in the access log file. Similarly in the RBAC software, there is a limit of 1000 subjects and 1000 accesses in the access log file. These limits can easily be extended if necessary by changing the space allocated to vectors in the program.

Chapter 8 – Evaluation

8.1 Introduction

In this chapter, the results of applying the evaluation criteria, specified in section 1.3, are reported. Following this some suggestions are made as to how the project could be extended upon in the future and finally an overall conclusion is presented on the project.

8.2 Results of Applying the Evaluation Criteria

8.2.1 Evaluating the Objectives and Minimum Requirements

Has the project achieved the objectives that have been set out?

Through the consultation of several textbooks and through reading many of the original articles associated with the security models, objective 1 was achieved, which was to research and investigate various security models. Furthermore, through the diagrams and the comparisons presented in chapter 3, an understanding of the way the security models work and differ has been demonstrated and has thus satisfied objective 2. This objective was also achieved through the design of metaphors in chapter 4 as this demonstrated a deeper understanding of the security models. Objective 3, which was to research SQL Server's security model and compare it to the other security models, was achieved through studying several articles published by Microsoft and through SQL Server's documentation. The research is presented in section 3.3.

Objective 4 was to research appropriate programming languages to use in the project. This objective was satisfied through the research carried out in section 2.3, where Java3D and OpenGL were investigated. In carrying out this research several web sources and textbooks were consulted. Objective 5 was to research development methodologies that may be used. To achieve this objective various methodologies were researched and the two most relevant methodologies have been documented in section 2.2. To carry out this research several project management textbooks were consulted as well as some lecture notes from a project management module.

Objective 6 was one of the major objectives of the project and was to design graphical metaphor(s) based on the security models. This objective was achieved through the various metaphors that were designed in chapter 4. Objectives 7 and 8, which were to design, implement, test and evaluate a piece of software have also been achieved. The software that has been implemented for the software has achieved this objective. The content of chapters 5, 6 and 7 document the design, implementation and testing stages of the software development. The evaluation of the software is carried out later on in this chapter (see sections 8.2.2-8.2.3 below). (Note: The full list of project objectives is in the summary section at the start of this document).

Has the project achieved the minimum requirements or have they been exceeded?

With reference to the minimum requirements set out in section 1.2:

- In chapter 3, four different security models were investigated, thus requirement 1, which stated that at least two security models should be investigated, was achieved and exceeded.
- The research on SQL Server's security model, presented in section 3.3, satisfied requirement 2, which stated that SQL Server's security model should be investigated and compared to the other security models.
- Requirement 3 stated that a graphical metaphor should be devised to represent one security model. This was satisfied through the metaphors developed in chapter 4. This requirement was actually exceeded as many metaphors were developed covering two security models.
- Requirement 4 stated that software should be designed and developed covering at least one security model. This requirement was exceeded as two pieces of software covering two security models were designed and implemented. Furthermore, the software was also produced to a much higher quality than first intended.
- Requirement 5 was to test and evaluate the software that was produced. The testing documented in section 7.3 satisfied part of this requirement. The evaluation part of this requirement was satisfied through sections 8.2.2 and 8.2.3 below.
- Further to this, some software documentation was also produced. This was used mainly to aid users taking part in the evaluation process, but was seen as an enhancement to the project. This documentation is presented in appendix E.

8.2.2 Evaluating the Software

Was the choice of programming language and development methodology appropriate?

After developing the software it seems that the choice of programming language was appropriate. This is because the implementation was carried out quite easily with very few problems due to the familiarity with the programming language. However had Java3D been used, then there may have been more problems, as it would have time taken to learn the language and also debugging and general programming would have been more difficult due to unfamiliarity. Furthermore, the programs that were developed contained a large amount of processing, and such amounts of processing may have slowed a Java program down considerably due to it running through a virtual machine. Also the choice to use GLUI was justified as it was easy to use and was found to be quite stable with few bugs.

The decision to use the "waterfall model with feedback" (see section 2.2) was also the right option as it allowed the software to be developed gradually over a number of iterations. It also allowed the program to be demonstrated at project meetings and at the progress meeting. Also, because it was developed over iterations, the testing and evaluation was carried out gradually throughout the implementation process. The nature of the methodology also allowed changes in direction, at the end

of each iteration. These things would not have been possible with the conventional waterfall model as it does not use iterations and uses an approach that relies on everything being right first time.

Is the software neatly coded and extensible?

The software has been coded as neatly as possible and extensive comments have been included in the code to make the code as understandable as possible. The code has been decomposed into functions where necessary and an attempt has been made to make the structure of the program as easily understandable as possible. It was possible to make the software more extensible by using a more object-oriented approach by using genuine classes rather than simply using C 'structs' as pseudo-classes. However this was not essential and it was easier to implement the program using structs. Also, the extra overhead involved in passing objects to and from functions may have slowed down the software and since it wasn't necessary to use classes, it was decided to use structs.

Does the software work and provide enough functionality?

The software was free from any major bugs. However there were some problems, which have been highlighted in section 7.2. The biggest problem with the software in terms of "bugs" is the window resize problem, which was explained in section 7.2. Other than this there weren't any major problems.

The software for the BLP model provided more than enough functionality to represent the main properties of the model. Although the software could be extended even further to represent for example the discretionary security policy discussed in section 3.2.1. The RBAC software on the other hand could be extended a lot further. While the software does enough to help users understand the main concept of the RBAC model i.e. that of roles, the software could still incorporate more and represent more of the RBAC models security rules. While such an extensive system has not been implemented, a design has been produce for such a system in appendix C.

8.2.3 Evaluation of User Feedback

For this section of the evaluation, the intention was to get some users to use the software to help them understand the security models the software represents. From this it was hoped that some feedback would be obtained to evaluate how well the software achieves its purpose. However, while this was the plan, in reality it was difficult to find people who were willing to take part in the evaluation process. This was mainly because it was a fairly lengthy process and most students were very busy. It may also have been because few people had the motivation to take part, as there are currently no specific security modules being taught to undergraduates in the School of Computing. Therefore only a few users took part in the evaluation, and many of these seemed to reluctantly agree to participate. This was disappointing as the evaluation may have been better if more users had taken part, but nevertheless some useful feedback was still obtained.

Does the software help people to understand the security model?

Overall feedback suggested that the software was very useful in helping users to understand the security models. Most users made the comment that the metaphors were fairly easy to look at and were certainly easier to study than formal descriptions of the model. Some specific comments made about each piece of software are discussed below.

For the Bell-LaPadula (BLP) software, most users felt that the layered architecture helped them to see how the security levels worked. However one person mentioned that they would prefer to see a more 3D version of the layers, as it may be easier to see the partial ordering through 3-dimensions. Generally users also commented that the metaphor made it easier to see how the BLP model combined the concept of layers together with the access control matrix. One user commented that the metaphor was probably "easier on the eye" than an access control matrix representing the same thing.

As for the properties of the BLP model, users said they understood what a "no-read-up" was after using the software. This showed that the users understood the ss-property of the BLP model. All but one of the users made the comment that the "no-write-down" property was still difficult to understand, although they found that they understood it better than they did before they used the software. They made the comment that even though the software showed them where a no-write-down was occurring, they found it difficult to understand why this was the case. This was probably because the no-write-down (*-property) is genuinely difficult to understand, and it would probably take more than a short evaluation session to understand exactly what the *-property represented. It has been assumed that if a student was genuinely using the software to learn about the BLP model then over a slightly longer period they would probably understand what the *-property represented.

For the Role Based Access Control (RBAC) software, people found that the metaphor was very easy to understand. In fact most people commented that the metaphor was easier to understand than the metaphor of layers used for the BLP model. This is probably because rooms are more familiar to users than layers. Most users seemed to understand how the contained roles worked and how this formed a hierarchy of roles. They also understood how the rights for each role were formed from the combination of the roles own rights and the rights of contained roles. Users also mentioned that the software gave them an easy way to see exactly what they'd allowed a user to do simply by looking at how the appearance of the rooms changed after they carried out an operation. However, a common comment made by users was that the functionality included to create accesses didn't really add to their understanding of the model. This was probably because the software represents a model where the majority of security is handled through the management of roles rather than through the accesses a subject makes, as is the case for the BLP model. This functionality was included to show how an access may be represented, as this may be necessary if the software was integrated into a real system.

Can users correctly identify what the software/metaphors represent and what is happening as the software runs?

For this section each item of software will be discussed separately. First, the BLP software will be discussed. Users found it easy to spot when an access was made and who made it. Users could also easily identify what accesses a subject was allowed to make both according to the subject's clearance level as well as according to the access control matrix. These were all easy for users to identify as they were clearly colour coded and once the different colours had been explained it was very simple to see what was happening. Users also found it easy to see when a violation of the ss-property (no-read-up) had occurred as if the access was in a layer that had been greyed out then it was a violation of the ss-property. However, at the early stages of development, it was found that it was extremely difficult to spot a violation of the *-property (no-write-down). Again, as explained before, this was due to it being a difficult concept to understand. Therefore, a feature was built into the software to make accesses which would cause a violation of the *-property to flash and also to draw a cross over any access that could potentially lead to a violation of the *-property. After this was introduced, users found that they could see the occurrence of a no-write-down more easily.

For the RBAC software, the users were asked to identify slightly different concepts. Users generally were able to easily identify what access a subject was making (read/write/none) and what object they were accessing. Users also found it very easy to identify exactly what rights a user was allowed through each role and when asked to describe the rights that were displayed on screen they usually found it trivial to do so. Users also seemed to understand the effects of activating/deactivating roles and could also point out how the metaphor of rooms might change if certain rights were added or removed. However, users didn't seem to have a proper understanding of what the effects would be of changes to the role hierarchy. I'm not quite sure why this was, but it may have been due to the fact that when the role hierarchy changed, many different aspects were affected, and therefore the users didn't quite understand exactly what impact the change would have.

Overall, users again found that the metaphor of rooms was a lot easier to understand than the metaphor of layers. This may suggest that the metaphor of rooms was a more natural fit with the users mental model than the concept of layers. However users did not seem to have much trouble with the BLP metaphor either, so it could be argued that users were able to find some link for this using their mental model as well. Generally, users faced difficulties with the more difficult ideas of the model. For example, in the BLP software users got confused due to the no-write-down policy and with the concept of partial orderings. With the RBAC model, they didn't face as many difficulties, as I believe the concepts of the model represented in the software are not that complex. However, had all the metaphors designed in section 4.4.2.2 been incorporated into the RBAC software, (as shown in appendix C), the users may have had more difficulties with the more complex concepts and with the way the metaphors changed as the users used the software.

Do users think the software is easy to use?

In general the users said they were comfortable with the interface and understood what most of the controls represented and what functionality they provided. However, they felt that some parts of the interface for the RBAC software were a bit cramped and felt that this made the interface a bit harder to understand. Another comment that was made was that at times they felt there were too many controls and they didn't quite know what to modify or what to press.

Generally users said that the interface was familiar with the use of drop down lists, checkboxes and radio buttons. They felt that while the interface was a bit complex at times it took a bit of getting used to. The number of control may have overawed users because they weren't actually motivated to learn the security models. As mentioned before, if a user was actually trying to learn the model of his/her own accord, rather than being convinced to do it, they may be more interested and may take more time to learn about the controls. One comment that was made by many users was that they preferred to use the interface provided in the software to interactively define things such as accesses, access rights etc. rather than use files as they didn't feel that using files was convenient.

8.3 Directions For Future Work

There are a number of directions that could be taken from the work produced in this project. This was originally an open-ended project and therefore there are almost limitless options that could be taken to enhance and build upon the work produced. In this section a few of these options are explained: -

- There are still parts of the software that are hard coded and not user modifiable (see section 7.4). These things could be worked on to allow the software to be more flexible for use.
- Some users mentioned that the metaphors would benefit from being 3-dimensional. Therefore it may be a good idea to attempt to convert the metaphors from two to three dimensions.
- The RBAC software could be regarded as being incomplete as not all the metaphors designed have been integrated. This is an enhancement that could be made. A design for this has been proposed in Appendix C.
- Other security models could be incorporated into the software that has already been produced. A possible starting point for this could be to read section 4.4.3 of this report.
- The software could be extended so that it could be integrated into a live system, where the access rights or the accesses apply to actual objects and where users could interactively access files and see the metaphor represent the accesses they are making.

8.4 Overall Conclusions

In general I am pleased with most of the work done during the project. I am happy with the security models that were investigated and feel that they were well understood. I am also happy with the

metaphors I have developed during the project. I feel this was a difficult task, which required independent thought and I feel the metaphors that were developed were original and represented the concepts and ideas of the security models quite well.

I also feel the software that was developed was produced to a high standard. While the evaluation has highlighted that the software still has some weaknesses, I feel that it would be a valuable aid to a student wishing to learn about security models. I also believe that while the software may not be suitable for immediate use, but the ideas developed during the course of the project may be valuable.

The one thing I was disappointed with was that I did not have enough time in this project to further develop both pieces of software. By this I mean that I would have liked to incorporate more features into both the pieces of software. This is true more so for the RBAC software than the BLP software. I would have liked to integrate all the other metaphors that were developed for this security model into the software but unfortunately there just wasn't enough time in a 40-credit project to do so.

Overall I feel that I am happy with the way the project has gone. I feel that all the objectives of the project were satisfied. Also the minimum requirements for the project were achieved and were extended upon quite considerably in some cases. In general I feel that the project met its aim in that it resulted in two pieces of software and several metaphors, which helped people to enhance their understanding of security models.

References

[1] Hughes, B & Cotterell, M, *Software project management,* McGraw-Hill, c2002

[2] Jesty, P, *'Life Cycle' notes*, **SO22 Lecture notes & handouts**, 2002-2003, URL: http://www.comp.leeds.ac.uk/so22/LIFECYCLES.PDF [25th April 2003].

[3] Sun Microsystems, *The Java 3D™ API: For Developers and End-Users*, URL: http://java.sun.com/products/java-media/3D/collateral/presentation/index.html [6th December 2002].

[4] Sun Microsystems, Technical White Paper, *The Java3D API,* Section 2, URL: http://java.sun.com/products/java-media/3D/collateral/j3d_api/j3d_api_1.html [6th December 2002].

[5] OpenGL Architecture Review Board, Woo, M, Neider, J, Davis, T & Shreiner D *OpenGL Programming Guide, The Official Guide To Learning OpenGL, Version 1.2.*, Addison-Wesley 1999.

[6] Kilgard M J, *The OpenGL Utility Toolkit (GLUT) Programming Interface API Version 3,* Silicon Graphics, Inc. URL: http://www.opengl.org/developers/documentation/glut/spec3/spec3.html [25th April 2003]

[7] Rademacher, P, *GLUI User Interface Library: Version 2.1 – User Manual,* URL: http://gd.tuwien.ac.at/hci/glui/ [25th April 2003]

[8] Azuma, D, (2000), *The GLOW Toolkit*, URL: http://glow.sourceforge.net/index.html [10th March 2002]

[9] Lampson, B, *Protection*, **ACM Operating Systems Reviews**, 1974 URL: http://research.microsoft.com/~lampson/09-Protection/WebPage.html [13th March 2003]

[10] Bell, D, & LaPadula, L, *MITRE technical report 2547 (secure computer system),* **Journal of Computer Society**, 4(2/3):293-263, 1996. Originally published as *Secure Computer Systems: Mathematical Foundations and Model*, The MITRE Corp., 1973 URL: http://www.mitre.org/resources/centers/infosec/secure_computers/ [25th April 2003]

[11] Dieter Gollmann, *Computer Security*, John Wiley & Sons, 1999

[12] Caelli, W, Longley, D & Shain M, *Information security handbook,* Macmillan, 1991.

[13] Harrison, M A, Ruzzo, M L & Ullman J D, *Protection in Operating Systems*, **Communications of the ACM**, 1975
URL: http://portal.acm.org/citation.cfm?id=806517&coll=portal&dl=ACM&ret=1#Fulltext [25th April 2003]

[14] Castano, S, Fugini, G M & Martella, G, *Database Security*, Addison-Wesley, 1994.

[15] Brewer, D F C, & Nash, M J, *The Chinese Wall Security Policy*. In **Proceedings of the 1989 IEEE Symposium on Security & Privacy**, 1989
URL: http://www.gammassl.co.uk/topics/chinesewall.html [25th April 2003]

[16] Ferraiolo, D. F., Cugini, J. A. and Kuhn, R. A., *Role Based Access Control: Features and Motivations*, **Computer Security Applications Conference**, 1995
URL: http://csrc.nist.gov/rbac/ [6th December 2002]

[17] Sandhu, R, Coyne, E, Feinstein, H & Youman, C, *Role-Based Access Control Models*, **IEEE Computer**, Volume 29, Number 2, February 1996, pages 38-47.
URL: http://citeseer.nj.nec.com/sandhu96rolebased.html [28th April 2003]

[18] Microsoft , (October 2000) *SQL Server 2000 Security White Paper,* pp 13-33
URL: http://www.microsoft.com/sql/techinfo/administration/2000/2000SecurityWP.doc [28th November 2002]

[19] *SQL Server Books Online,* provided with Microsoft® SQL Server™ 2000

[20] Hosmer H., *Visualising Risks: Icons for Information Attack Scenarios,* **23rd NISSC Proceedings**, 2000, pp: 4-5,
URL: http://csrc.nist.gov/nissc/2000/proceedings/papers/050.pdf [28th January 2003]

[21] Wilson D, Rauch T, and Paige J, *Generating Metaphors for Graphical User Interfaces,* 1998. URL: http://www.firelily.com/opinions/metaphor.html [28th January 2003]

[22] Ruddle, R, *User Interface Design, Lecture 2 notes*, **SI13 – Human Computer Interaction Lecture notes & handouts**, 2000-2001,

Appendix A – Personal Reflections

Looking back over the course of the project, I would have to say that the project experience was a difficult yet enjoyable one. This project was a fairly large task and involved a large amount of research, innovative design and software engineering, but I feel that I have coped fairly well.

The project got off to a good start. I enjoyed researching security models and designing metaphors. These stages took up a large chunk of the early part of the project and went according to plan and were highly enjoyable. I also enjoyed the freedom to work at my own pace. I found that managing my own time over such a large project was an invaluable experience. Furthermore, I was able to plan my time in a way so that I was never overawed by the schedule but also I always had something to do.

There were also parts of the project that didn't go quite as well as planned. While I enjoyed managing my own project, I found that I underestimated certain stages of the project and thus some stages took slightly longer than planned. I also did not compensate for busy periods when planning my project and this led to problems when there were a lot of coursework deadlines. Another stage of my project that did not go to plan was the implementation phase. At the early stages, I had designed everything quite well before I started to code anything. But, in the latter stages of implementation, I found that I was spending less time designing and rushed into the coding. This was due to the incremental nature of the methodology used and others using similar methodologies should be aware of this problem.

From the experiences of this project I have learned to plan everything thoroughly before carrying it out. This especially applies to software coding. Also, I learnt that when planning a schedule for the project, some time should be set aside to allow the project to catch up, otherwise the project could fall behind schedule. I have also learned that when implementing software, you must decide on a clear stopping point. Otherwise you will keep extending on the software and other stages will fall behind.

To future students, I would provide the following advice. Firstly, when you choose a project ensure you choose one that you like and one that you are interested in as this will motivate you to spend time on the project rather than put it to one side. Also, try to document any progress you make as you go through your project. This will make it a lot easier when it comes to writing up the project report. Furthermore, if possible, try to write up parts of your project as you go along, as this reduces the pressure of having to write the entire report at the end of the project.

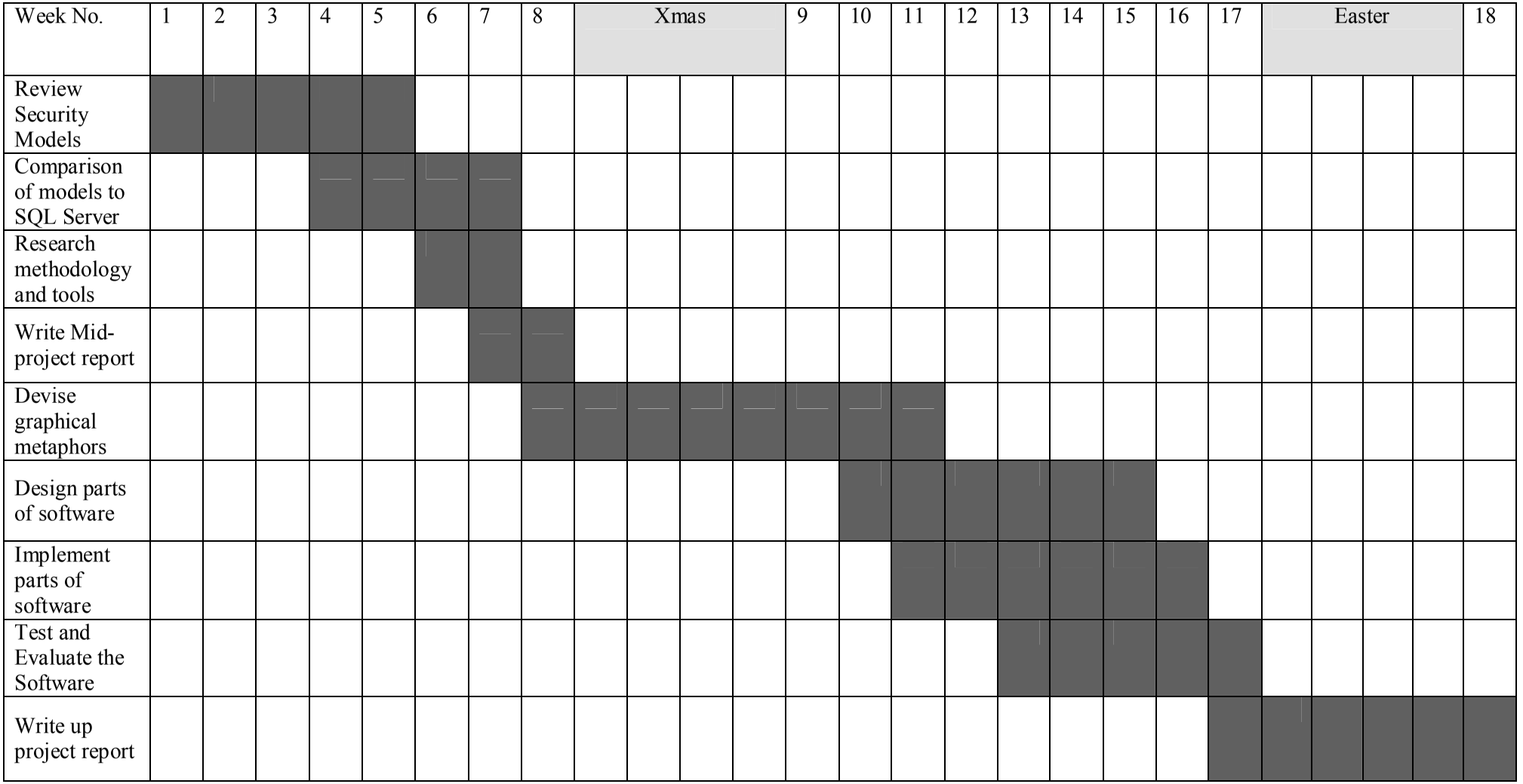
Overall I am happy with the work that has been produced in the project. If I did the project again, I would try to learn from the lessons highlighted above and would attempt to start implementing the software earlier as I believe I could have improved the RBAC software further if I had the time.

Appendix B – Project Schedules

Schedule Version 1 - This is the original version of the project schedule drawn up at the start of the project.

Week No.	1	2	3	4	5	6	7	8	Xmas	9	10	11	12	13	14	15	16	17	Easter	18
Review Security Models																				
Comparison to SQL Server																				
Devise graphical metaphors																				
Design Software																				
Implement Software																				
Evaluate Software																				
Write up project report																				

Schedule Version 2 - This is the second version of the project schedule. This was produced late in the first semester when, due to sickness and coursework deadlines, I fell behind the original schedule. Therefore some of the stages were put back a bit further to run until after Christmas.



Schedule Version 3 – This version of the project schedule was drawn up between Christmas and Easter in the second semester. This was required due to a period when there was a large amount of coursework to be submitted which led to the project falling behind schedule.

Week No.	1	2	3	4	5	6	7	8	Xmas				9	10	11	12	13	14	15	16	17	Easter				18
Review Security Models																										
Comparison of models to SQL Server																										
Research methodology and tools																										
Write Mid-project report																										
Devise graphical metaphors																										
Design parts of software																										
Implement parts of software																										
Test and Evaluate the Software																										
Write up project report																										

Appendix C – Potential Design For RBAC Software

In this section, a provisional design is presented to allow all the metaphors designed for the RBAC model, to be integrated into one application. Figures C.1, C.2 and C.3 below show the different aspects of the interface design. The following paragraphs explain the figures.

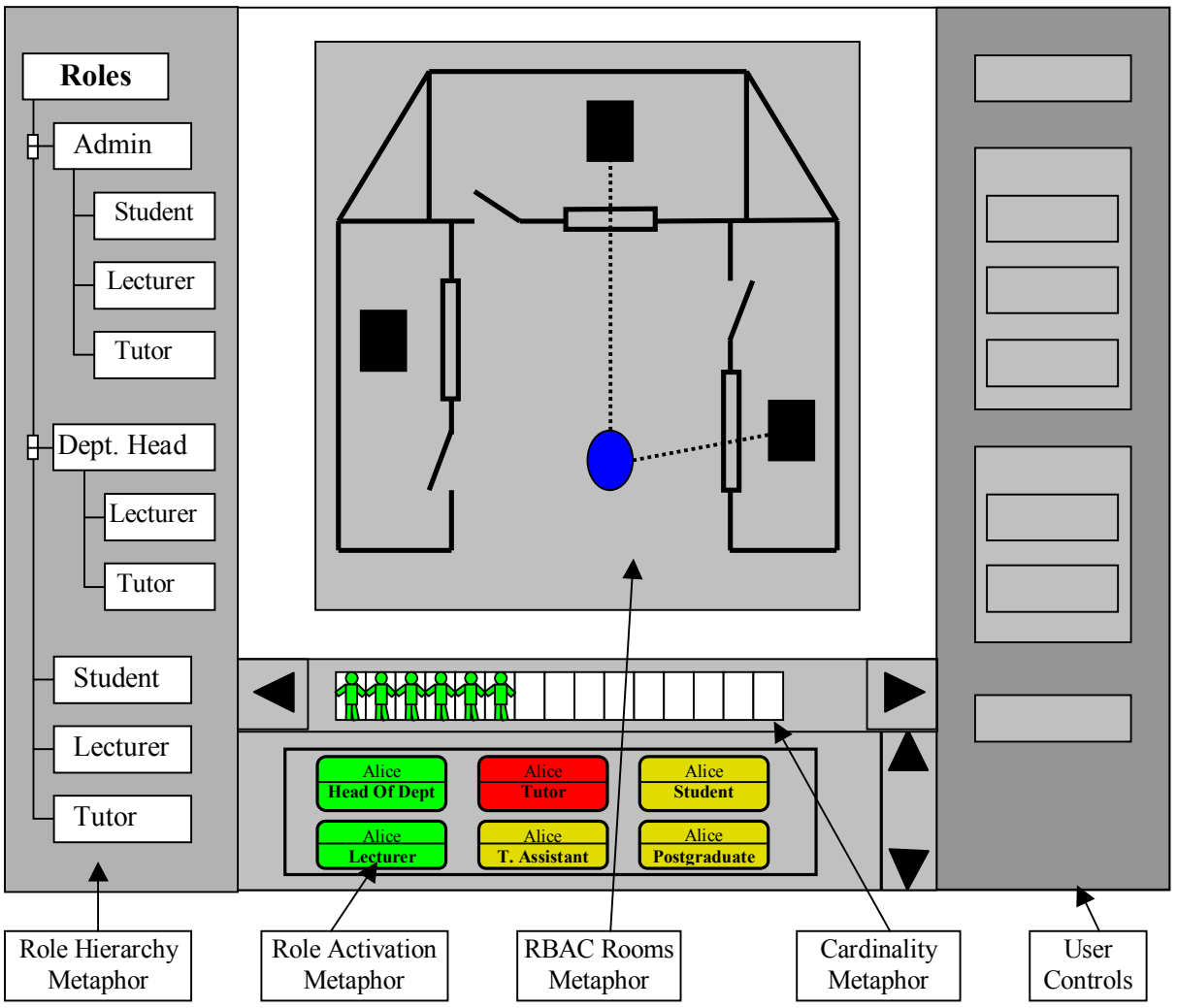


Figure C.1: RBAC Model Software – Interface Design

Figure C.1 above shows the main window of the program. This design shows that the main display will incorporate several of the metaphors designed for this model. The display shows that the metaphor of rooms will be at the centre of the display. The user controls are docked to the side of the window. These controls will be similar to those described in section 5.3.2. To the right of the display, the metaphor for the role hierarchy will be displayed. This will display the roles and could even allow the user to change the role hierarchy by moving roles about in the metaphor. The window also shows the cardinality metaphor. This metaphor will be displayed under the rooms metaphor and the user will be able to see how many users are active in a role (the specific role could be selected by the user or

the metaphor could be scrollable to show different roles). The role activation metaphor will also be displayed in the main window and will show what roles the currently viewed subject is authorised for, which ones are active and which ones are mutually exclusive.

Figure C.2 (right), shows the design for the popup window which would allow users to authorise roles for a subject. This window will show the roles that have been authorised and the roles that have not yet been authorised. The user would then be allowed to select the roles for activation from the 'not authorised' box and either drag them across or click a button to say this role is authorised. Obvioulsy, greyed out roles would not be allowed to be authorised.

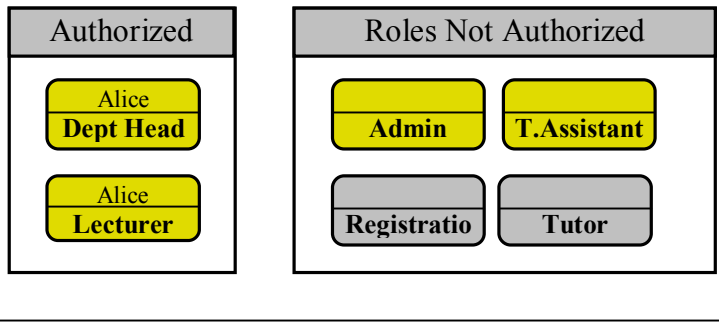


Figure C.2: Popup window to authorise roles for subjects

Figure C.3 below, shows the popup window that would allow users to modify the rights for each role. There would be a convential interface to allow the rights to be defined (shown to the left of the diagram). However, there would also be a display showing the metaphor for operational separation of duty. This would only show the metaphor for operations where the operational separation of duty constraint has been defined. This would allow users to see whether or not the operational separation of duty constraint has been violated or not.

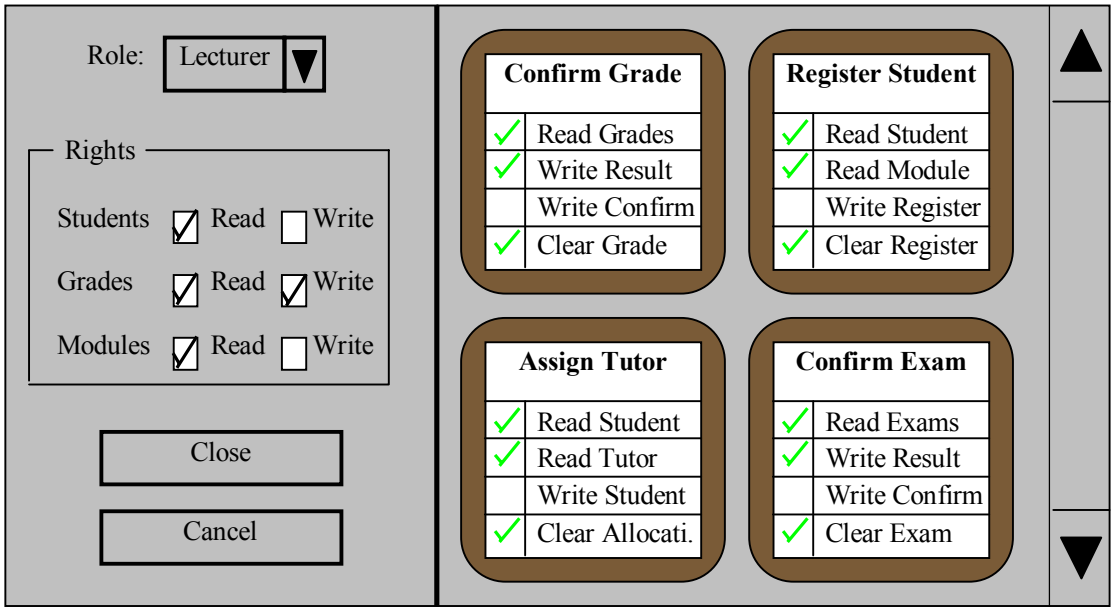


Figure C.3: Popup window to change access rights for each role

Appendix D – Test Plans

Test Plan For BLP Software

Function	Test To Be Applied
Reading from files	Does it work with an empty file? Does it stop correctly at end of file?
Drawing Objects	Are objects drawn the right colour? Are labels drawn correctly? Are restrictions according to no-write-down rule drawn correctly? Are objects drawn correctly when they are accessed?
Drawing Layers	Are layers drawn the right colour? Are the objects drawn in the correct position and in the correct layer? Does the layer allow proper scrolling to view different objects?
Drawing The Access Log Display	Does it draw the correct accesses at the correct times? Does it scroll properly and allow previous accesses to be viewed?
User Interface	Does changing the currently viewed subject work as expected? Does creating accesses work for different subject/object/access combinations? Does pausing and stepping the simulation work? Does downgrading of levels option work as expected? Does "write includes read" option work as expected?

Test Plan For RBAC Software

Function	Test To Be Applied
Reading from files	Same as for BLP software (see above)
Drawing Rooms	Are windows & doors on rooms drawn correctly for different access rights? Are all the rooms joined together correctly & are correct messages displayed?
Drawing Subject	Is subject drawn in right position/orientation for different scenarios?
Drawing Read Lines	Are the read lines drawn correctly? i.e. to correct room at correct times.
Drawing Access Log	Same as for BLP software (see above)
User Interface	Does changing the currently viewed subject & role work as expected? Do the "activate role" & "deactivate role" buttons work as expected? Does creating accesses work for different subject/role/object combinations? Does pausing and stepping the simulation work? Does changing "contained" roles work as expected for different scenarios? Does changing role access rights work as expected for different scenarios? Does changing authorised roles work as expected for different scenarios?

Appendix E – Software Documentation

Bell-LaPadula (BLP) Model Software

Reading From Files – This software reads in details from four files. Each of the files should be terminated with a –1 on a new line. The details of the four files are explained below: -

File Name: subjects **Schema:** subjectID, subjectName, clearance
Explanation – This file defines the subjects to be placed into the system. The subject ID is the unique identifier for the subject. This should start at 0 and increment for each subject in the file. The subject name is simply the name of the subject and should be a single string. The clearance is the clearance level of the subject and is in the form of a level ID. The levels span from 1 to 4 (with 1 = Public layer (lowest), 2 = Programmer layer, 3 = Designer layer and 4 = Manager layer (highest)).

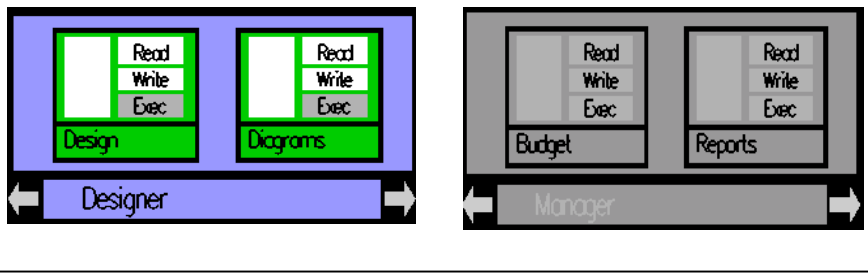
File Name: objects **Schema:** objectID, objectName, classification
Explanation – This file defines the objects to be placed into the system. The object ID is the unique identifier for the object. This should start at 0 and increment for each object in the file. The object name is the name of the object and should be a single string. The classification is the classification level of the subject and is in the form of a level ID. The levels span from 1 to 4 as for the subjects file.

File Name: rights **Schema:** subjectID, objectID, rights
Explanation – This file defines the rights of the subjects on the objects. The subject ID refers to a subject defined in the subject file. The object ID refers to an object defined in the object file. The rights are defined as three chars i.e. rwx. These three chars refer to the rights the subject, referred to through subjectID, is allowed on the object, referred to through objectID. The first char represents the right to read, the second the right to write and the third the right to execute the object. If a read is allowed you put an ‘r’ in otherwise you put a ‘-‘. The same applies for write and execute except using ‘w’ and ‘x’ respectively. There should not be any spaces between each of these chars e.g. if read and execute were allowed you would put r-x in the file.

File Name: accesses **Schema:** subjectID, objectID, access
Explanation – The subject ID refers to a subject defined in the subject file. The object ID refers to an object defined in the object file. The access refers to the access made and similarly to the rights file is defined using three chars i.e. rwx. These three chars refer to the read, write and execute access. So as explained before for the rights, a similar concept is used for the accesses made. If you want to define a subject reading a file then you define the ID’s appropriately then you put r—in the file to show only a read is taking place. This file can be left empty with a –1 if you want to use the interactive menu to define accesses whilst the software is running rather than define them in advance.

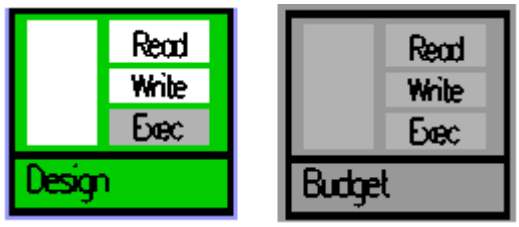
The software – This section describes what the different things displayed on the metaphor actually mean and what the different colours and icons represent.

Layers – The layers are represented as either authorised or unauthorised. Authorised layers are shown in blue and unauthorised layers are shown in grey.

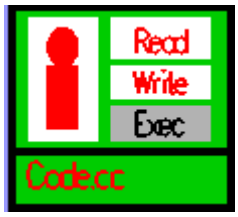


Authorised and Unauthorised Layers

Objects – Objects are drawn as authorised or unauthorised as well. Authorised objects are drawn green and unauthorised ones are drawn grey. If a user is authorised to read an object then the read label on the object is displayed whit but if not then it is greyed out the same applies for write and execute as well. The image below left shows this. The green object has read and write operations authorised for it. Also if an object is accessed then the object's labels are highlighted red. So if an object is being read from, then the read label turns red. This is shown in the right image below.



Authorised and Unauthorised Objects



An object being accessed

No-Write-Downs – If a no-write-down occurs in the program then this is also indicated through the metaphor. If a subject accesses an object causing a no-write-down, then the offending right will flash in the metaphor. This is shown in the image below left. However, accesses that could potentially cause a no-write-down are also be marked with a read cross. This is shown in the image below right.

A no-write-down flashing

A read that may cause a no-write-down

User Controls – The control panel is shown in the image below. These controls are now explained.

Subject Being Viewed -> Subject – This allows you to change the currently viewed subject so another subject is displayed.

Add A New Access – This menu allows you to define a new access or to release a current access. You must select a subject and an object from the drop down lists and then you must tick the relevant boxes to define the type of access. Finally when you have made your choices click on the add access button to confirm the access and display it.

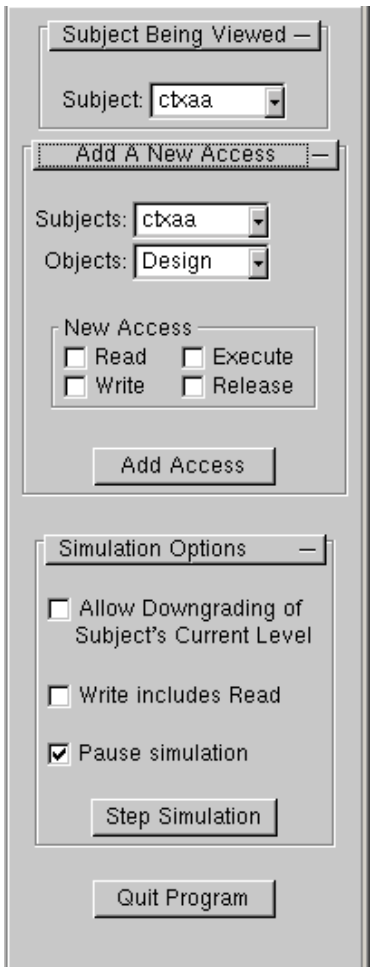
Simulation Options

Allow Downgrading Of Subjects Current Level – This option determines whether or not to allow the subjects current level to be downgraded. If this is ticked then subjects are allowed to write to files below their clearance. However, if this is not ticked, then any write below a subject's clearance level is seen as a no-write-down.

Write Includes Read – This option determines whether writes are treated as blind writes or writes that allow the subject to read the object at the same time. If this is not ticked then writes are regarded as blind writes and this will apply to every write in the system.

Pause Simulation – Pauses the simulation so no more accesses are read in from the access log.

Step Simulation – Steps the simulation so the next access is read in from the access log.



User control panel

Quit Program – Quits the program.

Role Based Access Control (RBAC) Model Software

Reading From Files – This software reads in details from six files. Each of the files should be terminated with a –1 on a new line. The details of the six files are explained below: -

File Name: subjects **Schema:** subjectID, subjectName
Explanation – This file defines the subjects to be placed into the system. The subject ID is the unique identifier for the subject. This should start at 0 and increment for each subject in the file. The subject name is simply the name of the subject and should be a single string.

File Name: roles **Schema:** roleID, roleName
Explanation – This file defines the roles available to subjects in the system. The role ID is the unique identifier for the role. This should start at 0 and increment for each role in the file. The role name is the name of the role and should be a single string.

File Name: userRoles **Schema:** subjectID, roleID
Explanation – This file defines the roles authorised for each subject. The subjectID refers to a subject defined in the subject file and the role ID refers to a role defined in the role file.

File Name: roleRights **Schema:** roleID, objectID, readRight, writeRight
Explanation – This file defines the rights of the roles on the objects. The role ID refers to a role defined in the role file. The object ID refers to an object in the system. The object ID ranges from 0 to 5 to represent the different objects hard coded into the system. The table to the right shows what object each ID represents. The readRight is defined as either r or –. The letter r signifies that the role is allowed read access on the object and the – signifies it is not allowed. The same applies for writeRight except the letter w is used. Note there should be a space between the readRight & writeRight.

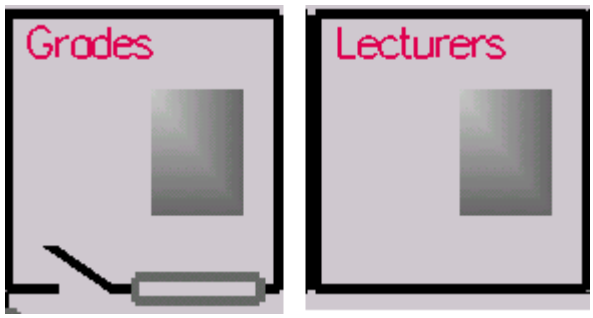
ID	Object Name
0	Students
1	Modules
2	Grades
3	Lecturers
4	Registrations
5	Tutors

File Name: contains **Schema:** parentRoleID, inheritedRoleID
Explanation – This file defines the role hierarchy. The parentRoleID and inheritedRoleID refer to roles defined in the roles file. So if you put in "1 2" then you are saying role 1 contains role 2.

File Name: accessLog **Schema:** subjectID, roleID, access, objectID(s)
Explanation – The subject ID refers to a subject defined in the subject file. The role ID refers to a role defined in the roles file. The access can be "activate", "deactivate", "read" or "write". If the access is anything but a read then one object ID between 0 and 5 must be defined. However if the access is a read then more than one object ID can be defined. This defines an access made. This file can be left empty with a –1 if you want to use the interactive menu to define accesses.

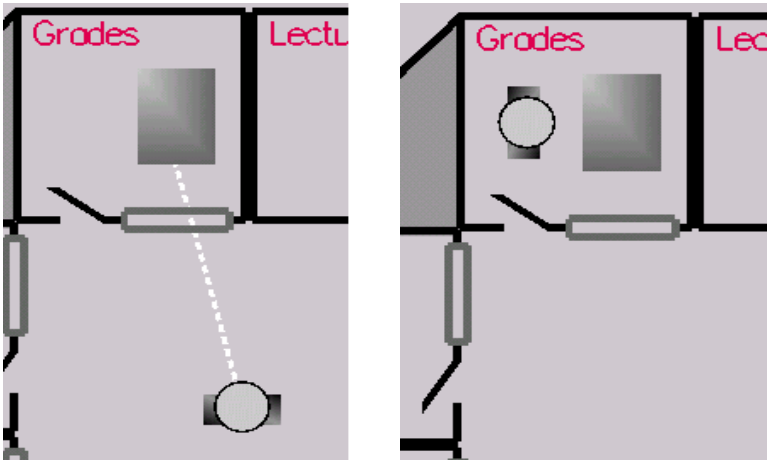
The software – This section describes what the different things displayed on the metaphor actually mean and what the different icons represent.

Objects Represented as Rooms – All the objects are represented as rooms and can have doors and windows. If a subject, through their active role, is allowed read access to an object then the room representing that object will have a window drawn on it. The same concept applies for write access but a door is used instead. If read access isn't allowed then a window isn't drawn. If write access isn't allowed then a door isn't drawn. The image below left shows a room where both read and write access are allowed and the image below right shows a room where neither read nor write access are allowed.



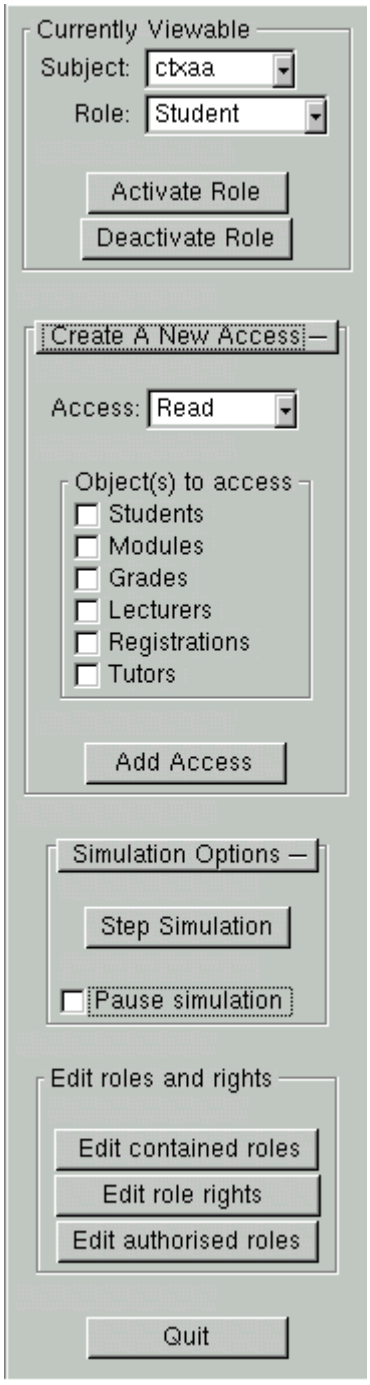
Rooms representing objects

Accessing Objects – When a subject accesses an object, the position of the subject may change and lines may appear. If white lines appear going from the subject icon to a room, then it indicates that the subject is reading from the object to which the line leads. This is displayed as shown in the image below left. There may be several lines as the subject may be reading from more than one object. If there aren't any lines but the subject icon has actually moved inside a room, then this indicates that the subject is writing to the object inside which the icon has moved. This is displayed as shown in the image below right.



Subject reading from (left) and writing to (right) an object (room)

User Controls – The control panel is shown in the image below. These controls are now explained.



Currently Viewable

Subject – This allows you to change the currently viewed subject so another subject is displayed through the metaphor.

Role – This allows you to change the currently viewed role so another role is displayed. This role is displayed for the subject selected in the drop down box above it.

Create A New Access – This menu allows you to define a new access. The new access will be created for the subject and role combination defined in the previous drop down box. First you select the type of access you want to create. Then you tick the relevant boxes to define the objects that are being accessed. When you have made your choices, click on the add access button to confirm the access and display it.

Simulation Options

Pause Simulation – Pauses the simulation so no more accesses are read in from the access log until the simulation is restarted.

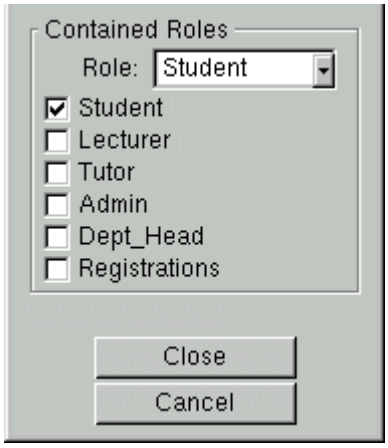
Step Simulation – Steps the simulation so the next access is read in from the access log when this button is clicked.

Edit roles and rights – These buttons launch floating windows, which allow you to do various things. These controls are explained on the following page.

Quit Program – Quits the program.

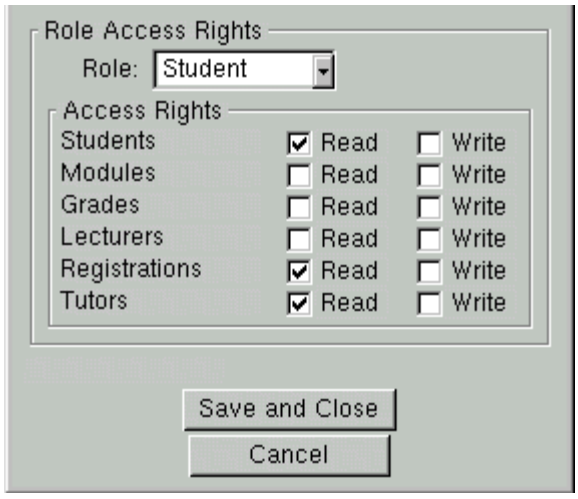
Edit roles and rights

Edit Contained Roles Window – This window allows you to change the role hierarchy by defining the roles that are contained within each role. If you want to contain one role in another, then you must first select the parent role from the drop down box. Once this is selected, the tick boxes will have ticks in the roles that are contained within the parent role (note the role will always contain itself and this cannot be changed). You can change the roles it contains by adding or removing ticks from the boxes. If you want to save your changes you must click on close. But if you don't want to save your changes click on cancel.



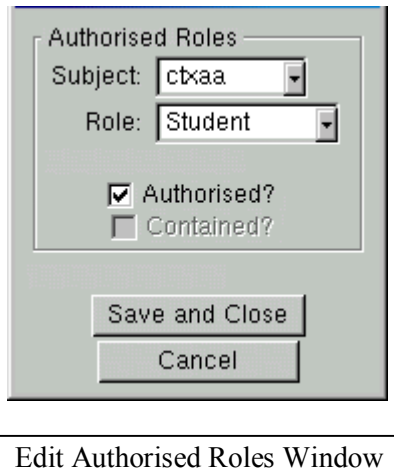
Edit Contained Roles Window

Edit Role Rights Window – This window allows you to change the rights of each role on the objects. First you have to select the role whose rights you want to change from the drop down box. Once this is done, the selected role's rights will be displayed by the checkboxes. You can now manipulate the role's rights by adding/removing ticks from these checkboxes. Note that these rights are the rights of the role only and not any rights that may have been inherited from contained roles. Again the close and cancel buttons work as explained before.



Edit Role Rights Window

Edit Authorised Roles Window – This window allows you to change the roles a subject is authorised for (i.e. allowed to activate). To do this you must first select the subject and the role you which to authorise from the drop down boxes. When this is done, the authorised box will show whether or not the user is already authorised for this role and the contained box will show whether the user is allowed to activate this role through another of the roles the user is authorised for. If you wish to authorise or unauthorise the user for this role, then you must tick/untick the box. The contained checkbox is for visual purposes and is not modifiable. The close/cancel buttons work as explained before.



Edit Authorised Roles Window